НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
КАФЕДРА АВТОМАТИКИ ТА УПРАВЛІННЯ В ТЕХНІЧНИХ СИСТЕМАХ

«На правах рукопису»
УДК _____

«До захисту допущено»
Завідувач кафедри
_____ _____
(підпис) (ініціали, прізвище)
“___”_____20__ р.

Магістерська дисертація

зі спеціальності (спеціалізації) 121, інженерія програмного забезпечення (інженерія програмного забезпечення комп’ютерних систем)_____

на тему: ___«Система захищенного зберігання персональних даних»_____

Виконав (-ла): студент (-ка) __2__ курсу, групи _____ІТ-83мп_____
(шифр групи)

Любченко Юрій Миколайович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник _____Полторак В.П. _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

РЕФЕРАТ

Магістерська дисертація складається з 112 сторінок, 15 зображень, 32 таблиць та 21 посилань на використані джерела.

Актуальність даної дисертації полягає в тому, що стандарт GDPR є новим, та на момент створення системи не існує аналогів, що дозволяють автоматизувати вимоги стандарту.

Мета дисертації – розробка системи для обробки персональних даних, що відповідають стандарту Європейського Союзу про збереження персональних даних.

Об'єктом дослідження дисертації є робота систем у відповідності до стандартів захисту даних.

Предметом дослідження є система, що дозволяє адаптувати інструментарій по збереженню та обробці персональних даних.

Під час вирішення завдань було застосовано загальні методи наукового пізнання, такі як: вимірювання та порівняння, спостереження, а також методи синтезу та аналізу. За допомогою вказаних наукових методів було досліджено існуючі рішення, та визначено підходи до оптимізації.

Результати роботи можуть бути використаними у якості інструментарію, що дозволяє додати підтримки сучасного стандарту захисту персональних даних до будь-якої системи.

Ключові слова: архітектура, контейнер, сервіс, односторінковий додаток, захист даних, веб-застосунок.

SUMMARY

The master's thesis consists of 112 pages, 15 images, 32 tables and 21 references to the sources used.

The relevance of this dissertation is that the GDPR standard is new, and at the time of creation of the system there are no analogues to automate the requirements of the standard.

The purpose of the thesis is to develop a system for processing personal data that meets the European Union standard on personal data retention.

The object of the research is the work of systems in accordance with data protection standards.

The subject of the study is a system that allows you to adapt the tools for storing and processing personal data.

Common methods of scientific knowledge, such as: measurement and comparison, observation, and methods of synthesis and analysis, were applied in solving the problems. Using these scientific methods, existing solutions were explored and optimization approaches identified.

The results can be used as a tool to add support for a modern standard of protection of personal data to any system.

Keywords: architecture, container, service, one-page application, data protection, web application.

Національний технічний університет України

«Київський політехнічний інститут
імені Ігоря Сікорського»

Факультет (інститут) Інформатики та обчислювальної техніки_____

(повна назва)

Кафедра Автоматики та управління в технічних системах_____

(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною (освітньо-професійною) програмою

Спеціальність (спеціалізація) 121 інженерія програмного забезпечення комп'ютерних систем

(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О. І. Ролік
(підпис) (ініціали, прізвище)
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Любченко Юрію Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації «Система захищеного зберігання персональних даних»

науковий керівник дисертації Полторак Вадим Петрович, к.т.н, доцент ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 20__ р. №____

2. Строк подання студентом дисертації

3. Об'єкт дослідження підтримка європейського стандарту захищеності персональних даних

4. Предмет дослідження (вихідні дані для магістерської дисертації за освітньо-професійною програмою) стандарт захисту персональних даних GDPR

5. Перелік завдань, які потрібно розробити ознайомлення із стандартом збереження персональних даних, проаналізувати існуючі рішення та виявити їхні сильні та слабкі сторони, сформувати вимоги до системи та сценарії використання, автоматизувати роботу з даними згідно до стандарту

6. Орієнтовний перелік ілюстративного (графічного) матеріалу діаграма компонентів, діаграма маршрутизації системи, діаграма аріхтектури, діаграма переходів, діаграма прецедентів

7. Орієнтовний перелік публікацій

8. Консультанти розділів дисертації[1*]

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання 24 жовтня 2018

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Порівняльний аналіз існуючих рішень	02.09.19 – 06.09.19	
2	Визначення основних вимог до системи	09.09.19 – 13.09.19	
3	Розроблення сценаріїв використання системи	16.09.19 – 20.09.19	
4	Вибір технологій для розробки	23.09.19 – 27.09.19	
5	Розроблення структурної схеми	30.09.19 – 04.10.19	
6	Розроблення ER-діаграми	07.10.19 – 11.10.19	
7	Реалізація бізнес-логіки	14.10.19 – 18.10.19	
8	Розроблення інтерфейсу користувача	21.10.19 – 25.10.19	
9	Розроблення стартап-проекту	28.10.19 – 02.11.19	
10	Оформлення текстового матеріалу	04.11.19 – 25.11.19	
11	Оформлення графічного матеріалу	26.11.19 – 02.12.19	
12	Подача дисертації на перевірку	03.12.19 – 18.12.19	

Студент _____ Ю.М. Любченко
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ В.П. Полторак
(підпис) (ініціали, прізвище)

[1*] Консультантом не може бути зазначено наукового керівника

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

- ІП - інформаційні процеси;
- GDPR - General Data Protection Regulation;
- XML - eXtensible Markup Language;
- MVC - Model-View-Controller;
- SOLID - single responsibility, open-closed, Liskov substitution, interface segregation и dependency inversion;
- ORM - object related mapper;
- JSON - JavaScript Object Notation;
- SQL - Structured Query Language;
- SPA - Single Page Application;
- DOM - Document Object Model;
- Аутентифікація - процес перевірки особистості;
- UI - User Interface.

ВСТУП

Сучасне суспільство все більшою мірою спирається на Інформаційні процеси, які стають рушійною силою економіки, суспільних відносин, військової справи [1]. ІП - це процеси збору, підготовки, передачі, обробки, перетворення та використання інформації в різних сферах суспільства. Інформація, за одним із продуктивних визначень, є корисні, або ж, нові відомості про навколишній і внутрішній світ.

Основними класифікаторами систем що зберігають інформацію є: цілісність, конфіденційність, доступність. А отже й основні класифікатори загроз поділяються на загрози цілісності, такі як знищення або модифікація; загроза доступності, це може бути блокування, або знищення інформації; загроза конфіденційності, насамперед це несанкціонований доступ, витік даних та розголошення.

Найбільш поширеним стандартом захисту даних на території Європи є GDPR [2], що регулює будь які дії з персональними даними користувачів, та автоматично вступає в дію, якщо хоча б один користувач є громадянином європейського союзу та є користувачем системи.

Основними принципами GDPR є:

- легальні підстави, в рамках стандарту, для збору та використання даних, не порушення будь-яких законів, відкрита та чесна інформація щодо використання цих даних;
- конкретні цілі зберігання даних, що повинні бути закріплені в політиці конфіденційності та повинні бути чітко дотримані;
- використання адекватної кількості даних, потрібних для виконання поставленої мети, які обмежені саме потрібною кількістю;
- персональні дані повинні бути точні, та не повинні вводити в оману;
- не зберігати дані довше, ніж це потрібно.

Вже створено багато систем, що частково або повністю підтримують цей стандарт зберігання даних, проте немає жодної реалізації, що допомогла би інтегрувати функціонал що задовольняє вимоги GDPR.

Цілями створення системи є:

- забезпечення можливості автоматичного видалення персональних даних за вимогою користувача;
- автоматична генерація умов використання даних, які система зберігає про користувача;
- забезпечення можливості автоматичного видалення персональних даних після вичерпання часу на їх використання щодо регламенту;
- вивантаження даних про користувача (при запиті);
- захист збережених даних;

Ключові завдання, які вирішуються завдяки розробці дисертації та складових системи адміністрування персональних даних:

- проаналізувати можливі типи архітектури системи та вибрати рішення, яке найкраще підходить для розробки розподіленої системи;
- спроектувати та розробити елементи системи для обробки запитів пов’язаних з наданням, зміною, видаленням та зчитуванням персональних даних;
- розробити відповідний інтерфейс користувача;
- визначити можливість використання елементів системи у підприємницькій діяльності.

Система є актуальною тому, що не має аналогів та вирішує актуальну проблему щодо захисту даних. Основною метою є надання простого інструменту, що допоможе дотримуватись європейського стандарту зберігання даних GDPR.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Інформація - це відомості, що є об'єктом зберігання, передавання та перетворення [3]. Захист інформації - це комплекс заходів технічного, організаційного та організаційно-технічного характеру, спрямованих на захист відомостей, що відносяться до певної особи, або визначаються на підставі будь-якої інформації про неї. Тобто, це завжди актуальне питання у сфері інформаційних технологій.

1.1 Опис предметного середовища

Система захищеного зберігання персональних даних - це програмне забезпечення для реалізації операцій з видалення, запису та зберігання персональних даних, а також певних операцій, пов’язаних з цими процесами. Реалізована система повинна обслуговувати проблеми контролера щодо збереження та операцій з персональними даними.

Основі етапи робіт, щодо захисту персональних даних [4]:

- визначити всі ситуації, коли потрібно проводити обробку персональних даних;
- виділити бізнес-процеси, в яких обробляються персональні дані;
- вибрати обмежене число бізнес-процесів для проведення аналітики;
- визначити коло інформаційних систем і сукупність оброблюваних персональних даних;
- виробити заходи щодо зниження категорій оброблюваних персональних даних;
- підготувати технічне завдання по створенню необхідної системи захисту;
- провести уточнення класів системи, з подальшою підготовкою рекомендацій по використанню технічних засобів захисту персональних даних;
- спроектувати і впровадити систему захисту персональних даних;
- взяти згоди на обробку персональних даних з суб'єктів персональних даних;
- проводити контрольні заходи щодо виявлення порушень захисту персональних

даних.

На рівні компонентів системи подібні аналоги зазвичай базуються на веб застосуванні, що є контролюючим та містить у собі бізнес логіку предметної області додатку. Також розрізняють рівень збереження даних, зазвичай це:

- реляційна база даних, котра зберігає у собі чутливі дані, які потрібно: бекапувати, тримати у захищеному та зашифрованому вигляді, зеркалювати, відновлювати (за необхідності);
- не реляційна база даних: використовується зазвичай як брокер задач, балансувальник навантаження, для підвищення продуктивності за рахунок кешування даних, також потрібна для керування розкладом періодичних задач, таких, як видалення застарілих даних, розсилка листів тощо;
- веб інтерфейс, що дозволяє контролювати роботу системи за допомогою графічних компонентів;
- сервіс, що обробляє бізнес логіку системи, тобто саме зберігає та оперує даними;
- сервіс, що слідкує за виконанням періодичних задач, ставить їх в чергу та виставляє для них пріоритети.

1.2 Огляд існуючих рішень

Для того щоб дослідити існуючі рішення, було обрано системи, інформацію про які можна було б знайти у мережі інтернет; основними характеристиками вибору стали:

- наявність веб-сторінки, котра є візитівкою системи, що була використана для ознайомлення з її функціями та можливостями;
- наявна стаття, що описує способи використання системи, або стаття про її реалізацію.

1.2.1 Огляд системи Evernote

Evernote [5] - це платформа що має веб-застосунок (рисунок 1.1) а також клієнтські додатки для різних платформ, таких як: Windows, Linux, Android та IOS.

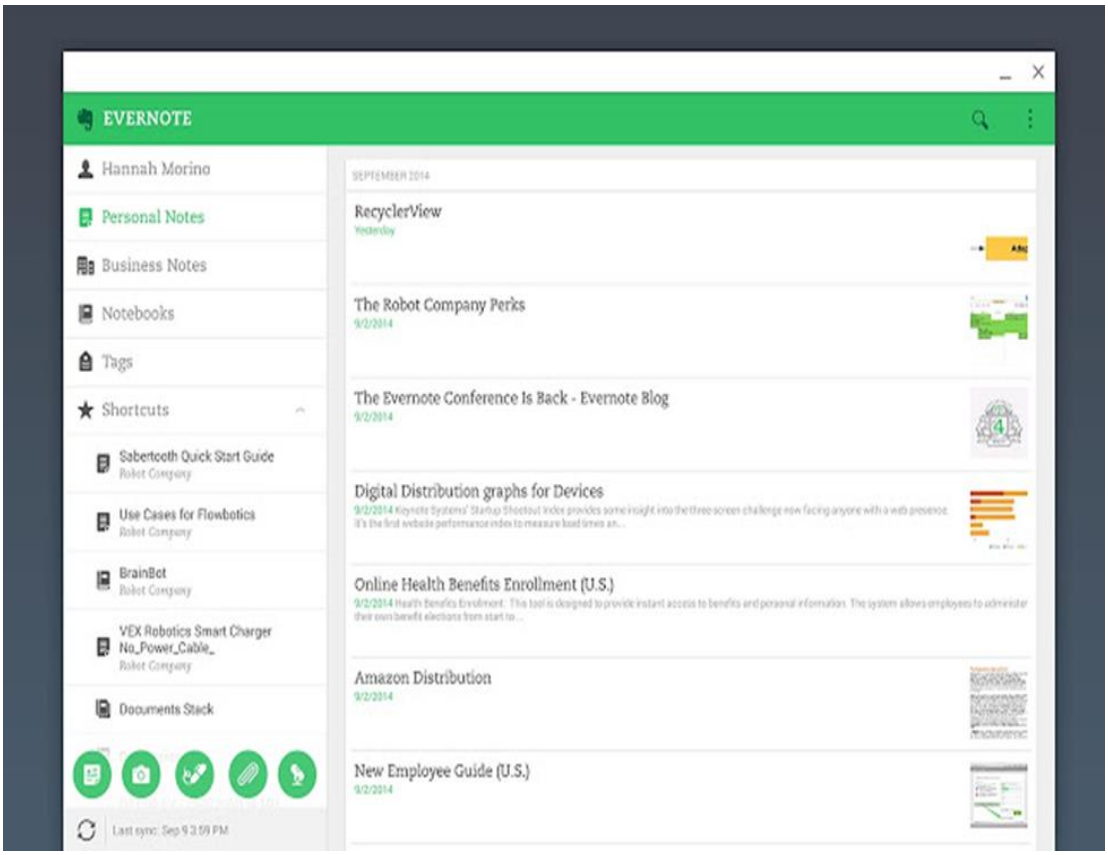

Рисунок 1.1 – Веб-сайт сервісу Evernote [5]

Основне призначення цієї платформи є пошук, збереження та відображення нотаток, зокрема веб-сторінок, рисунків, ключів, електронної пошти, іншої приватної інформації. Ця система має функціонал щодо запам'ятовування будь-яких видів інформації, від неважливої до більш значної та приватної. Система має вбудований веб-браузер для переглядання посилань та функцію попереднього перегляду збережених файлів.

Система позиціонує себе як сервіс для збереження всього, тобто є потужним інструментом що використовує велика кількість користувачів; вона має гнучкий, інтуїтивно зрозумілий інтерфейс, що має розширену можливість пошуку у структурах

документів функціями прикріплення файлів та зображень. Має безкоштовний профільний план, проте він обмежений у розмірі простору для зберігання, та за кількістю заміток.

Система має наступні функції:

- прикріплення інформації до запису;
- захист частини записів за допомогою вибраного паролю;
- редагування записів;
- пошук по вкладенням, тегам або ключовим словам;

Однак система має суттєві недоліки, а саме:

- обмежене виділене місце для збереження файлів у системі, а саме - 60 МБ;
- обмежена кількість можливих записів для безкоштовного плану;
- часткова невідповідність стандарту GDPR;
- розповсюдження аналітики про дані.

1.2.2 Накопичувач документів у смартфоні «ВКармане»

Сервіс [6] позиціонує себе як « гаманець у телефоні», тобто додаток, що дозволяє користуватись персональною інформацією, а саме збереженими зображеннями та документами, як дані, доступ до яких потрібно завжди мати при собі. Основною перевагою додатку є функционал що забезпечує збереження часу користувачів, що вони приділяють на пошук потрібного документу чи реквізиту.

Згідно з інформацією, що було опубліковано на офіційному сайті розробників, сервісом користуються приблизно 200 тисяч користувачів, а збережено у сервісі майже 500 тисяч записів та файлів.

Також розробники додали функцію експорту даних, що збережені, таких як документи, до інших додатків, що вимагають реквізити користувача. Цей функціонал автоматизує рутинну роботу на яку користувачі витрачають багато часу. Графічний інтерфейс (рисунок 1.2) орієнтований на різні за розміром та відношенням сторін

дисплеї, такі як комп’ютери, планшети та мобільні пристрої, забезпечує зручний та інтуїтивно зрозумілий доступ до інтерфейсу з кожного з них.

Основні функції сервісу:

- заповнення реквізитів та кредитних карток;
- експортування даних у сторонні додатки;
- доступ до файлів, що найчастіше використовуються без мережі інтернет;
- заповнення шаблонів існуючих документів;
- шифрування інформації;

Загальні недоліки сервісу:

- часткова підтримка GDPR;
- заблокований доступ до частини документів за відсутності інтернет з’єднання;
- небезпечне для збереження файлів в єдиному екземплярі, останнє оновлення видалило всі дані користувачів;
- низкий рейтинг застосунку на онлайн платформі App Store, 1.3 з 5 балів. Він різко впав через невдале оновлення, та знищення даних користувачів, крім того у відгуках зазначена інформація щодо неодноразових технічних збоїв.

Огляд сервісу, та загальний аналіз помилок і проблем, що виникають у системах, пов’язаних з чутливими персональними даними показує, що критерій, котрий повинен цінуватися в системі - це цілісність даних користувачив, це впливає на рейтинг та загальну оцінку системи серед користувачів.

1.2.3 Огляд системи MyTetra

MyTetra [7] - це система, що є кросплатформним рішенням, та використовується для накопичення інформації у вигляді заміток і статей. Записи організовані у вигляді деревоподібної структури, а також забезпечені ключовими словами, та іншими фільтрами за бібліотечною системою.

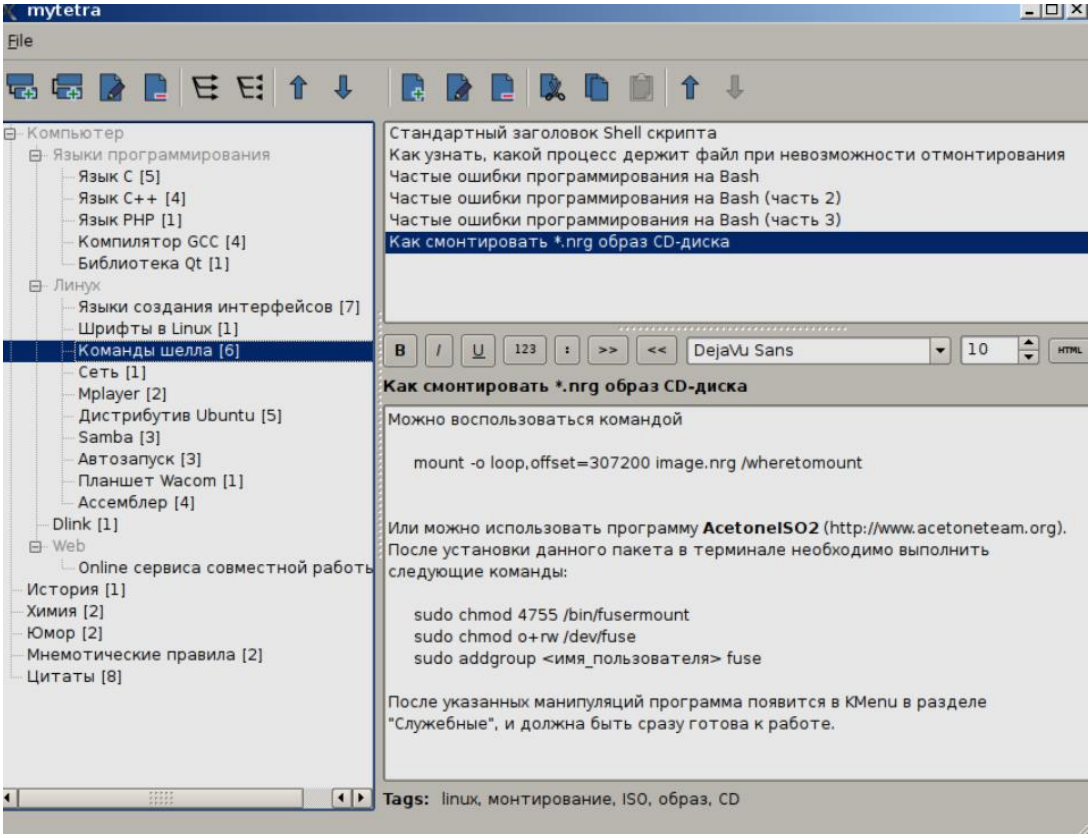

Рисунок 1.2 – Інтерфейс програми «MyTetra» [7]

MyTetra позиціонує себе, як програма , що має зручний та легкий в користуванні інтерфейс (рисунок 1.2),що реалізує зручний пошук по дереву та швидку навігацію. Дерево поділене на сутності “гілка” і “запис”; реалізований підрахунок кількості записів в гілках, копіювання записів через буфер обміну. Розробники програми відмовилися від зберігання даних в БД. Всі формати відкриті: дерево зберігається в XML-файлі, форматований текст-в HTML, картинки в PNG, а налаштування в INI.

Основні функції сервісу:

- навігація по дереву записів;
- редагування та видалення заміток користувача;
- пошук по тегам та ключовим словам;
- шифрування інформації.

Завдяки QT Only підходу програму можна запускати на всіх платформах.

Загальні недоліки сервісу:

- дані зберігаються на сторонньому хостингу для підтримки контролю версій «GitHub», що не підтримує GDPR;
- веб-клієнт є по суті JavaScript-сторінкою, і відображені на ній дані не індексуються пошуковими сервісами.

Найбільшою проблемою систем є зберігання приватних даних, адже у відкритому вигляді їх не можна завантажувати на хостинг. Розробниками була створениа невелика криптографічна бібліотека, і на її основі зроблено шифрування обраних гілок. Завдяки цьому приватні дані безпечно зберігаються у всіх перед очима, але ніяк не обробляються.

1.2.4 Система для бізнесу «ValkyrieCRM»

Система з включенням модулів управління контактами, персоналом, списком завдань, документами та подіями, що реалізує автоматизацію бізнес процесів [8]. Ця система підлаштовується до вимог кожного клієнта завдяки офіційному заключенню контракта між фірмою-розробником та клієнтом, інтегрується зі сторонніми платформами, такими як MailChimp, Twilio і SandGrid, створюючи єдиний призначений для користувача інтерфейс для осіб, які приймають рішення та членів команди. Функціонал доступний з браузера через веб-сайт (рисунок 1.3).

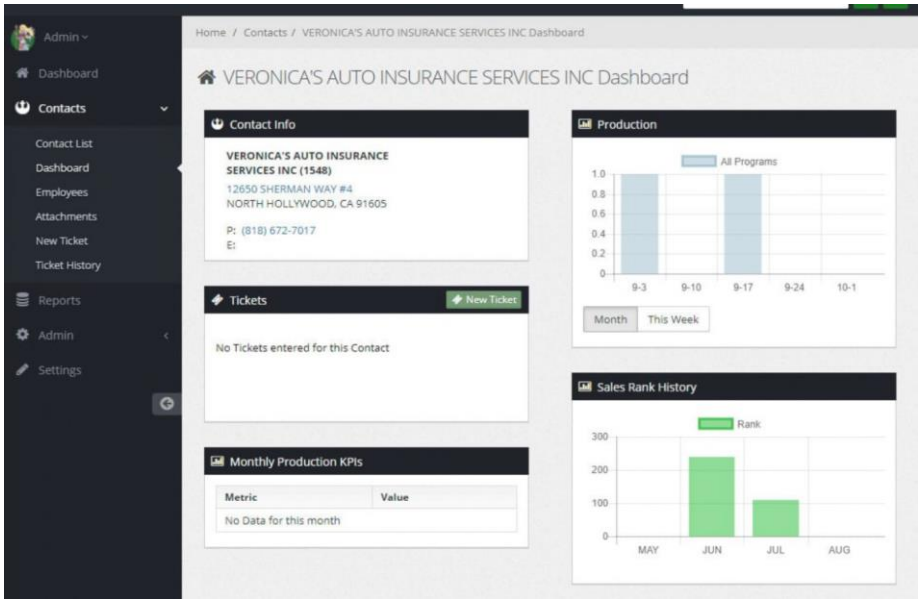

Рисунок 1.3 – Веб інтерфейс системи «ValkyrieCRM» [8]

Функціонал, що забезпечує панель адміністратора:

- перегляд відомостей;
- управління персональними даними працівників;
- автоматичне створення звітів;
- налаштування функціоналу під конкретні вимоги;
- аналіз активності клієнтів;

Недоліки даного програмного комплексу:

- впровадження вимагає контакту з компанією-розробником;
- потрібно багато часу для налаштування функціоналу під конкретні потреби замовника;
- часткова підтримка GDPR.

Продукт зберігає та використовує багато даних клієнтів та використовує їх номера телефонів та адреси електронних пошт для розсилок.

1.3 Висновки до розділу

У результаті огляду існуючих рішень, було визначено функції систем, їх загальні переваги та недоліки.

Основними функціями є :

- робота з створення , видалення та редагування записів;
- збереження файлів користувачів;
- захист інформації шифруванням чи іншим способом;
- структурна організація даних.

У рішеннях, що наведені в розділі було визначено наступні недоліки:

- часткове або повне недотримання стандарту GDPR;
- відсутність цілісності збережених даних;
- відсутність резервного копіювання даних;
- відсутня можливість масштабування потужностей та функціоналу системи;
- використання хмарних сховищ, як метод забезпечення збереженості даних.

Важливим недоліком розглянутих систем є спосіб збереження даних. Під час використання хостингів даних типу GitHub або BitBucket для збереження даних мінусом є правила хостингу про те, що розміщені там дані є відкритими для всіх під різними ліцензіями, а це є незадовільним фактором, відповідно до Європейського стандарту зберігання даних GDPR.

При обробці даних у великих обсягах є необхідність в оцінці ризикованості конкретних операцій або їх груп з персональними даними.

Метою дисертації є реалізація та автоматизація основних положень нормативного акту GDPR , таких як:

- отримання згоди суб'єкта персональних даних у вигляді окремого документа або в форматі окремого дії;
- забезпечення суб'єкту персональних даних можливістю відкликати дозвіл на використання даних в будь-який момент;
- забезпечення прозорих механізмів збору, обробки, зберігання, та використання персональних даних;
- забезпечення видалення персональних даних по першій вимозі власника таких

даних;

- своєчасне виявлення порушень прав суб'єкта персональних даних , а також термінове оповіщення останнього про таке порушення.

Таким чином, встановлена необхідність розробки системи що задовольняє отримані вимоги, для вирішення проблем щодо безпеки даних. Для проектування та подальшої розробки визначені основні питання, які потребують вирішення, а саме - повний доступ користувачів до управління своїми персональними даними, реалізований за допомогою гнучкої архітектури з можливістю доповнення системи новими модулями. Основною характеристикою системи встановлена - безпека та прозорість щодо збереження даних, тобто система повинна повністю інформувати користувача щодо дій з його даними, а також зберігати їх цілісність.

2 ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ

Аналіз вимог - одна з ключових частин процесу розробки програмного забезпечення, що містить: збір вимог до системи, систематизацію та розподілення, виявлення взаємозв'язків, а також документування.

Важливим фактором у процесі збору вимог є можливі суперечності, що виникають при розгляданні системи зі сторін різних зацікавлених осіб, такими можуть бути як розробники, так і замовники системи. Повнота та якість аналізу вимог є головним фактором успіху всього проекту.

Вимоги розрізняють на функціональні та нефункціональні. Аналіз вимог можна розділяти за трьома різновидами діяльності:

- збір вимог - аналіз предметної області, огляд існуючих рішень, спілкування з клієнтами, а також з майбутніми користувачами;
- аналіз вимог - визначення таких характеристик, що впливають на систему, як: цілісності, суперечності вимог між собою, взаємозв’язку між ними, та їх повноту;
- документування вимог - вимоги повинні бути задокументовані, це можна зробити у різних формах, таких як простий опис, специфікація процесів, або за допомогою сценаріїв використання.

2.1 Функціональні вимоги

Функціональна вимога - це опис поведінки системи відповідності до функціональністю системи, тобто описує що повинна робити система.

Основними функціональними вимогами є:

- система повинна бути гнучкою, працювати з будь-якою структурою бази, за умови, що потрібні сутності вказані у конфігураційному файлі;
- система повинна надати користувачу, чітку, цілісну, та чесну інформацію щодо видів даних, та часу впродовж якого вони будуть збережені у системі;
- система повинна за вимогою користувача надавати всі дані про нього, що

класифікуються системою як персональні;

- система повинна мати функцію, що в автоматичному режимі видаляє всі збережені у ній дані про користувача за його вимогою;
- система повинна зберігати дані у зашифрованому та захищеному сховищі, це ж стосується і бекапів, та файлів користувача.
- система повинна автоматично видаляти застарілі а також дані, що користувач позначив як видалені з усіх бекапів, при цьому можливий сценарій коли система не видаляє дані з бекапів проте робить це одразу ж після відновлення бекапу, проте це потрібно погодити з користувачем;
- система повинна чітко розділяти користувачів на ролі, та визначати їх права та обмеження;
- система повинна надавати можливість змінити, або видалити будь-які персональні дані про нього;
- система повинна надати користувачу форму погодження з політикою збереження даних та їх використання перед наданням доступу до функціоналу;
- система повинна автоматично видаляти дані про користувача з усіх інтегрованих систем в автоматичному режимі;
- застарілі резервні копії даних повинні бути видаленні з носіїв після завершення періоду їх актуальності.

2.2 Нефункціональні вимоги

Нефункціональні вимоги - це опис вимог, що конкретизують характеристики системи, такі як: доступність, потужність, відповідність, відновлюваність, ефективність, розширюваність, відмовостійкість, конфіденційність, якість, надійність, стійкість, масштабованість, безпечність, стабільність, підтримуваність та інші.

Основними функціональними вимогами є:

- дані повині резервно копіюватися раз на добу;

- резервні дані повинні мати період оновлення встановлений у розмірі 30 діб;
- за вимогою користувача дані про нього повинні бути видалені не пізніше ніж через добу після запиту;
- дані, що були визначені користувачем, як дані для видалення, повинні бути видалені з резервних копій одразу ж після відновлення, до початку продовження роботи системи;
- система може бути допущена до впровадження або оновлення лише після покриття кодової бази на вісімдесят або більше відсотків тестами;
- всі резеврні копії, файли, та дані повинні бути зашифровані;
- усі компоненти системі повинні бути подані у вигляді докер контейнерів, що забезпечить масштабованість та стійкість системи.

2.3 Висновки до розділу

У цьому розділі було сформульовано загальні функціональні та нефункціонавльні вимоги до системи. Вимоги до системи є ключовими характеристиками, що визначає якість системи, та її доцільність для користувачів. У відповідності з усіма типами вимог, пропуск тієї чи іншої вимоги може потенційно поставити під загрозу цілісність і повноту рішення. Функціональні і нефункціональні вимоги тісно пов'язані між собою множинними взаємозв'язками.

Кожне рішення досягає ефективності з вичерпного переліку вимог, зібраних як на початку, так і під час процесу впровадження. Вимоги можуть бути розділені на дві широкі категорії: істотні і фундаментальні. Істотні вимоги випливають з їх аналогії з «функціональними вимогами» і, як видається, мають безпосереднє відношення до вирішення. Однак так звані фундаментальні вимоги можуть не мати прямого зв'язку з рішенням, але вони мають основоположне значення для створення стійкого середовища, в якій зберігаються істотні функціональні вимоги. Тому ці «нефункціональні вимоги» складають структуру та інфраструктуру, які підтримують системні рішення.

Під час формування вимог до системи, було обрано загальні шляхи розвитку, такі як:

- масштабованість, якої було досягнуто за рахунок контейнеризації окремих компонентів системи;
- своєчасність проходження процесів щодо резервного копіювання, видалення, та представлення даних користувача;
- тестування, як головна характеристика якості та готовності системи, та її окремих компонентіва;
- захист даних, що було досягнуто завдяки шифруванню бази даних, файлового сховища, що зберігає файли користувача, та сховища що зберігає резервні копії бази даних;
- безпека доступу до даних, користувачі повинні мати чіткі ролі у системі та розмежований доступ до даних, відповідно до відведеної ролі.

3 СЦЕНАРІЇ ВИКОРИСТАННЯ СИСТЕМИ

Сценарій використання - це опис поведінки системи під час взаємодії з людиною або чимось з зовнішнього середовища, тобто відмінний інструмент для опису вимог таким чином, який цілком зрозумілий для користувачів і зацікавлених сторін і, в той же час, готовий до використання в команді розробки.

Методика сценарію використання використовується для формулювання вимог до поведінки системі щодо дій актора, таких як функціональні або визначені користувачем вимоги.

Актор - це роль, яку відіграє людина або річ, що взаємодіє із системою. Людина що використовує систему, може виступати як різні актори, оскільки вони відіграють у системі різні ролі.

Сценарії використання позначають систему як "чорну скриньку", а взаємодії з системою, такі як відповіді системи, описані термінами зовнішнього спостерігача. Цей метод побудови архітектури змушує зосередитися на тому, що повинна робити система, а не на тому, як це можливо зробити, і уникає припущень, щодо того як реалізовувати розглянутий функціонал.

Сценарії використання повинні бути описані на абстрактному рівні, та описані на рівні функцій системи та визначає функцію або послугу, яку система надає користувачеві, а також описує, що може зробити актор під час взаємодії з системою.

Опис проведено на рівні функцій системи та визначено функцію або послугу, яку система надає користувачеві, а також описує, що може зробити актор під час взаємодії з системою.

Сценарій використання повинен:

- процес, що повинна зробити система, аби актор досяг своєї мети;
- не використовувати детей реалізації;
- достатньо деталізувати процеси;
- не описувати екрани та інтерфейси що призначені для користувача.

На основі визначених вимог до системи - були розроблені і описані сценарії використання. Система має два головних актора для взаємодії з нею.

Користувач - отримує доступ до функціоналу, такого як перегляд та видалення всіх даних що збережені на нього, та перегляд аналітичної інформації. Процес аутентифікації користувача у системі проходить за допомогою звернення до неї через систему що виступає контролером даних. На основі визначених функціональних вимог, було розроблено та описано сценарії використання, що зображені на рисунку 3.1.

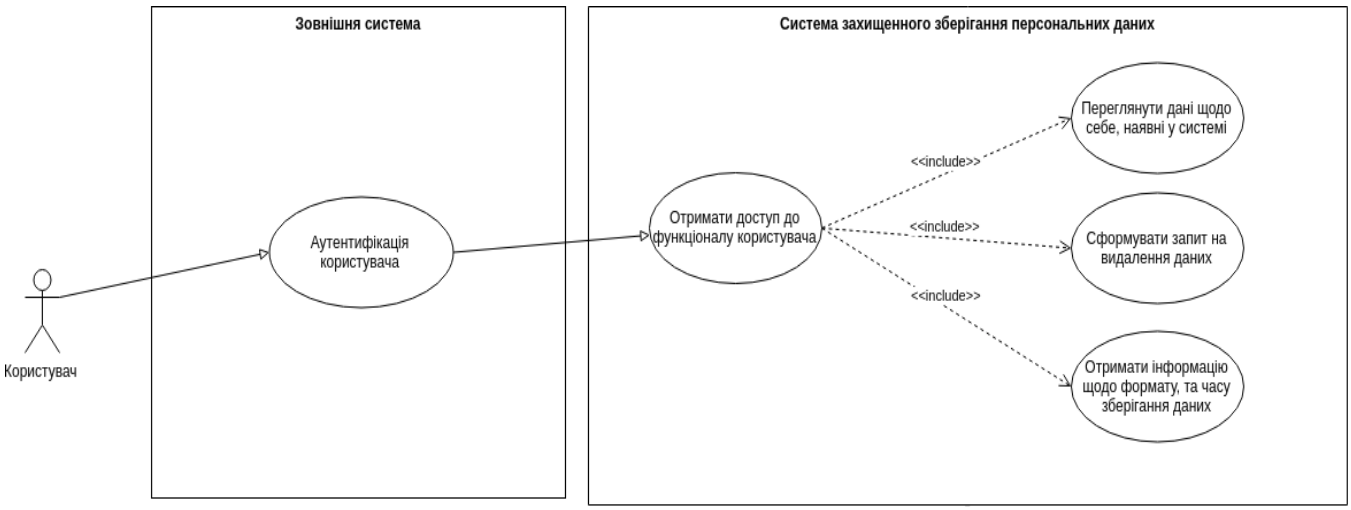

Рисунок 3.1 – Діаграма прецедентів для ролі “Користувач”

Адміністратор - актор, основна роль якого полягає у конфігурації системи. Основна функція ролі - виконання сервісних функцій, таких як створення незапланованої резервної копії даних, а також конфігурація схеми персональних даних у системі.

Адміністратор на відміну від користувача має доступ до графічного інтерфейсу системи, що дозволяє гнучко її конфігурувати.

Було розроблено та описано сценарії використання, що зображені на рисунку 3.2.

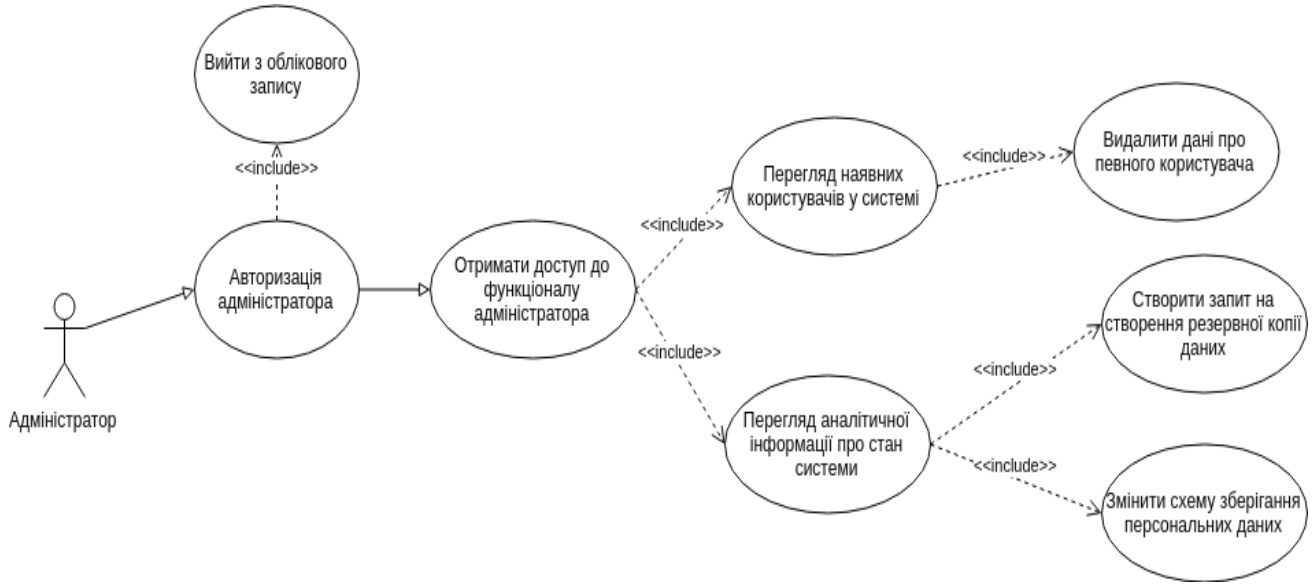

Рисунок 3.2 – Діаграма прецедентів для ролі “Адміністратор”

Детальний опис прецедентів зображений на таблицях 3.1 - 3.10, а повну діаграму представлено в додатку А.

Таблиця 3.1 – Опис прецеденту аутентифікації користувача.

Назва	Аутентифікація користувача
ID	1
Опис	Зовнішня система, що має ключ доступу, передає ідентифікатор користувача.
Актори	Користувач
Організаційні переваги	Користувач має доступ лише до своїх даних, та до функціоналу, що дозволяє їми оперувати.

Продовження таблиці 3.1

Частота використання	При кожному запиті від імені користувача
Причина виникнення	Користувач бажає отримати доступ до функціоналу системи,

	та оперувати власними даними
Передумова	Зовнішня система була успішно авторизована, та ідентифікатор користувача знайдений у базі.
Постумова	Користувача було успішно індефіковано в системі, доступ до функціоналу був надан.
Головний шлях виконання	1. Користувач бажає отримати доступ до функціоналу системи 2. Користувач відправляє запит про це до зовнішньої системи 3. Система формує запит, додає свій, попередньо сконфігурований ключ доступу, та ідентифікатор користувача. 4. У разі підтвердження, користувач ідентифікується у системі.
Альтернативний шлях виконання	Альтернативні шляхи виконання відсутні.
Виключення	При виникнені помилки під час аутентифікації, зовнішня система буде проінформована про це за допомогою статус кода у відповіді, а також допоміжного інформативного повідомлення щодо помилки.

Таблиця 3.2 - Опис прецеденту переглядання особистої інформації

Назва	Переглядання особистої інформації
ID	2
Опис	Користувач має змогу створити запис та отримати всі дані, та всі файли, що наявні на нього у системі.

Актори	Користувач
Організаційні переваги	Користувач має функціонал, що генерує взагалі всі збережені дані про нього у єдиний файл, це дуже зручно для аналізу, не обмежено можливостями відображення даних через графічний інтерфейс, та дозволяє використовувати системи аналізів логів, для інформування в агрегованому вигляді.
Частота використання	У випадках необхідності перевірки власних даних
Причина виникнення	Згідно до стандарту GDPR, користувач повинен мати змогу переглянути, змінити, та видалити всі збережені на нього дані що наявні у системі.
Передумова	Запит на отримання даних був сформований, користувач ідентифікований у системі.
Постумова	Користувач отримав дані у вигляді архіву, що містить об’єкт формату JSON з інформацією, а також його особисті файли, що були збережені у системі.

Продовження таблиці 3.2

Головний шлях виконання	1. Користувач бажає отримати доступ власних даних та файлів, що були збережені у системі 2. Користувач відправляє запит про це. 3. Система формує відповідь на запит, огортаючи обьект з даними та файли у архів. 4. У разі успішного виконання, система повертає посилання, за якою можно скачати створений архів.

Альтернативний шлях виконання	Альтернативні шляхи виконання відсутні.
Виключення	При виникненні виключення під час формування запиту, користувач отримає у відповідь статус код помилки, а також інформативне повідомлення, що містить у собі текстовий опис виключення.

Таблиця 3.3 - Опис прецеденту отримання інформації щодо формату даних.

Назва	Отримання інформації щодо формату даних, та кількості днів, на які вони можуть бути збережені.
ID	3
Опис	Довідка, представлена у вигляді JSON об'єкта, що надає користувачу інформацію про формат, за яким будуть збережені його персональні дані.

Продовження таблиці 3.3

Актори	Користувач.
Організаційні переваги	Користувач має змогу отримати інформацію щодо кожного рядка, у якому буде збережено його персональні дані. Ця інформація містить кількість днів до застарівання даних, тип застарівання (це може бути час з останнього використання, або ж час створення даних), довідкова інформація, щодо інформативності поля.
Частота	Після реєстрації у зовнішній системі, перед початком

використання	роботи, після кожної зміни формату, при запиті від користувача.
Причина виникнення	Згідно до стандартів GDPR, користувач потрібен мати змогу на перегляд формату, та визначеного часу застарівання його персональних даних у системі.
Передумова	Запит на отримання формату даних був сформований, користувач ідентифікований у системі.
Постумова	У разі успішного виконання, користувач отримає дані у вигляді об’єкту JSON.
Головний шлях виконання	1. Користувач відправляє запит про отримання формату даних. 2. Система формує відповідь на запит, та сериалізує відповідь у об’єкт. 3. У разі успішного виконання, система повертає об’єкт з інформацією щодо формату даних.

Продовження таблиці 3.3

Альтернативний шлях виконання	Альтернативні шляхи виконання відсутні.
Виключення	При виникненні виключення під час формування запиту, користувач отримає у відповідь статус код помилки, а також інформативне повідомлення, що містить у собі текстовий опис виключення.

Таблиця 3.4 - Опис прецеденту видалення даних.

Назва	Видалення даних користувача.

ID	4
Опис	Видалення або анонімізація усіх наявних даних про користувача у системі.
Актори	Користувач.
Організаційні переваги	Користувач має змогу створити запит на видалення усіх своїх даних з системи.
Частота використання	Одноразово, коли користувач хоче завершити назавжди роботу з системою.
Причина виникнення	Згідно до стандартів GDPR, користувач потрібен мати змогу на видалення власних даних з системи.
Передумова	Запит на видалення даних отриманий, користувач ідентифікований у системі.

Продовження таблиці 3.4

Постумова	Усі дані щодо користувача видалені, користувач отримав повідомлення про успішне виконання.
Головний шлях виконання	1. Користувач відправляє запит про видалення даних. 2. Система, згідно до файла з форматом, видаляє усі існуючі дані, та файли, що наявні. 3. У разі успішного виконання, система повертає інформативне повідомлення щодо виконання операції.
Альтернативний шлях виконання	Альтернативні шляхи виконання відсутні.
Виключення	При виникненні виключення під час видалення, користувач отримає у відповідь статус код помилки, а також

	інформативне повідомлення, що містить у собі текстовий опис виключення.

Таблиця 3.5 - Опис прецеденту авторизації адміністратора.

Назва	Авторизація адміністратора
ID	5
Опис	Авторизація адміністратора у системі за допомогою імейла та пароля.
Актори	Адміністратор
Організаційні переваги	Адміністратор зареєстрований в системі і має доступ до її функціоналу.

Продовження таблиці 3.5

Частота використання	Кожен раз, коли сесія з сервером не знайдена у системі, або не зареєстрована на клієнті.
Причина виникнення	Для розмежування доступу до персональних даних користувача, адміністратор проходить авторизацію та підтверджує свою особу.
Передумова	Адміністратор не має чинної сесії з сервером, та перейшов на сторінку авторизації.
Постумова	Адміністратор успішно здійснив авторизацію у системі, та отримав доступ до функціоналу системи.
Головний шлях виконання	1. Адміністратор переходить на сторінку авторизації. 2. Адміністратор вводить необхідні дані, та натискає кнопку підтвердження.

	3. Система перевіряє дані облікового запису, та у разі, коли вони знайдені успішно авторизує адміністратора у системі, перенаправляючи на сторінку перегляду аналітичної інформації.
Альтернативний шлях виконання	Альтернативні шляхи виконання відсутні.
Виключення	Можливі помилки під час заповнення або валідації форми, або помилка виконання доступу до бази тощо. Адміністратор системи буде сповіщений за допомогою вспливаючого діалогово вікна з кодом помилки, а також текстовим пояснювальним повідомленням.

Таблиця 3.6 - Опис прецеденту перегляду користувачів системи.

Назва	Перегляд користувачів системи
ID	6
Опис	Адміністратор має змогу переглянути список всіх наявних користувачів у системі
Актори	Адміністратор
Організаційні переваги	Адміністратор має змогу переглянути наявних користувачів у системі.
Частота використання	За наявної причини з відстедення підозрілого користувача, або з причини видалення користувача з системи.
Причина виникнення	Адміністратор повинен мати змогу переглядати існуючих користувачів.

Передумова	Запит на отримання сторінки з користувачами.
Постумова	Адміністратор успішно авторизований,
Головний шлях виконання	1. Сесія, та обліковий запис користувача знайдені у системі. 2. Користувач переходить на сторінку з відображеними користувачами у табличному форматі.
Альтернативний шлях виконання	Альтернативні шляхи виконання відсутні.

Продовження таблиці 3.6

Виключення	Можливі помилки перевірки сесії у системі, а також помилки підключення до бази даних. Адміністратор системи буде сповіщений за допомогою вспливаючого діалогово вікна з кодом помилки, а також текстовим пояснювальним повідомленням.

Таблиця 3.7 - Опис прецеденту видалення даних користувача.

Назва	Видалення даних користувача
ID	7
Опис	Видалення всіх даних та файлів користувача за допомогою функціоналу адміністратора
Актори	Адміністратор
Організаційні переваги	Адміністратор має змогу видаляти дані користувача згідно з запитом.

Частота використання	Лише після офіційного запита від користувача.
Причина виникнення	Користувач може не мати змоги видалити власні дані, проте через офіційний документ попросити зробити це через адміністратора системи.
Передумова	Запит на видалення користувача.

Продовження таблиці 3.7

Постумова	Користувач видалений з системі, адміністратор отримав повідомлення щодо успішно виконаного запита.
Головний шлях виконання	1. Адміністратор перебуває на сторінці перегляду даних користувачів, має активну сесію та обліковий запис. 2. Адміністратор створює запит на видалення певного користувача. 3. У разі успішного виконання, адміністратор отримує повідомлення про це. Користувач видалений із системи, та списку користувачів.
Альтернативний шлях виконання	Альтернативні шляхи виконання відсутні.
Виключення	Можливі помилки перевірки сесії у системі, а також помилки підключення до бази даних. Адміністратор системи буде сповіщений за допомогою вспливаючого діалогово вікна з кодом помилки, а також текстовим пояснювальним повідомленням.

Таблиця 3.8 - Опис прецеденту перегляду аналітичної інформації.

Назва	Перегляд аналітичної інформації
ID	8
Опис	Перегляд даних щодо кількості користувачів, останніх дій з видалення користувачів, формату даних, ротації резервних копій.

Продовження таблиці 3.8

Актори	Адміністратор
Організаційні переваги	Адміністратор має змогу переглядати всю актуальну аналітичну інформацію на одній сторінці, та слідкувати за станом системи..
Частота використання	Після успішної авторизації адміністратора, та за запитом адміністратора.
Причина виникнення	Адміністратор бажає слідкувати за роботою системи а також форматом збережених даних
Передумова	Адміністратор авторизований у системі, створив запит на отримання аналітичної інформації.
Постумова	Аналітичні дані відображені на сторінці
Головний шлях виконання	1. Адміністратор має активну сесію та обліковий запис. 2. Адміністратор створює запит на сервер, для отримання аналітичної інформації. 2. Інформація відображена на сторінці.
Альтернативний шлях виконання	Альтернативні шляхи виконання відсутні.

Виключення	Можливі помилки перевірки сесії у системі, а також помилки підключення до бази даних. Адміністратор системи буде сповіщений за допомогою вспливаючого діалогово вікна з кодом помилки, а також текстовим пояснювальним повідомленням.

Таблиця 3.9 - Опис прецеденту створення резервної копії.

Назва	Створення резервної копії
ID	9
Опис	Створення позапланової резервної копії даних
Актори	Адміністратор
Організаційні переваги	Під час оновлення чи переносу системи адміністратор має змогу позачергово створити резервну копію даних, що містить актуальну інформацію для подальшого відновлення.
Частота використання	При оновленні системи, чи проведенні технічних робот з системою.
Причина виникнення	Адміністратор повинен мати змогу позачергово зберегти резервну копію даних для підтримки їх актуальності.
Передумова	Адміністратор був успішно авторизований у системі, знаходиться на сторінці перегляду аналітичної інформації. Створив запит на створення резервної копії.
Постумова	Адміністратор отримав повідомлення у спливаючому вікні щодо успішної реєстрації задачі по створенню резервної копії у системі.
Головний	1. Адміністратор має активну сесію та обліковий запис.

шлях виконання	2. Адміністратор створює запит на створення резервної копії даних. 3. Створена задача, що виконується у фоновому режимі. 4. Адміністратор отримує повідомлення щодо успішного виконання.

Продовження таблиці 3.9

Альтернативний шлях виконання	Альтернативні шляхи виконання відсутні.
Виключення	Можливі помилки перевірки сесії у системі, а також помилки підключення до бази даних. Адміністратор системи буде сповіщений за допомогою вспливаючого діалогово вікна з кодом помилки, а також текстовим пояснювальним повідомленням.

Таблиця 3.10 - Опис прецеденту зміни формату даних.

Назва	Зміна формату даних
ID	10
Опис	Адміністратор має змогу змінити формат збереження даних, та час їх збереження.
Актори	Адміністратор
Організаційні переваги	Для зміни формату даних не потрібно мати додаткових навичок, достатньо використати веб інтерфейс застосунку.
Частота використання	При зміні структури даних у зовнішній системі.
Причина виникнення	Функція полу автоматичної адаптації формату даних при їх

	зміні у зовнішній системі.

Продовження таблиці 3.10

Передумова	Адміністратор був успішно авторизований у системі, знаходиться на сторінці перегляду аналітичної інформації. Створив запит на зміну формату даних.
Постумова	Запит успішно виконаний, формат даних змінений, користувачі отримали повідомлення щодо зміни формату даних.
Головний шлях виконання	1. Адміністратор має активну сесію та обліковий запис. 2. Адміністратор створює запит на зміну формату даних 3. Конфігураційний файл було змінено. 4. Таблиця з користувачами, що погодились з форматом даних була очищена. 5. Адміністратор отримав повідомлення щодо успішного виконання.
Альтернативний шлях виконання	Альтернативні шляхи виконання відсутні.
Виключення	Можливі помилки перевірки сесії у системі, а також помилки підключення до бази даних. Адміністратор системи буде сповіщений за допомогою вспливаючого діалогово вікна з кодом помилки, а також текстовим пояснювальним повідомленням.

Основними ролями, що наявні у системі було визначено адміністратора та користувача. На основі визначених функціональних вимог було описано сценарії використання.

Система має два способа авторизації. Перший, для користувачів, зовнішній, тобто зовнішня система авторизує користувача та індефікує його. Другий, для адміністраторів, за допомогою веб інтерфейса системи.

Користувач взаємодіє за допомогою REST API у форматі JSON, усі операції мають відповідь загального характеру з кодом успішності виконання, та повідомленням щодо виконання, або цільовою інформацією для користувача. Адміністратор взаємодіє за допомогою вбудованого веб інтерфейсу, що дає змогу налаштувати формат даних, переглянути аналітичну інформацію, видалити користувача, та зробити резервну копію даних.

4 СТРУКТУРНА СХЕМА СИСТЕМИ

Структурна схема відображає склад та взаємодію частин системи. Структурна схема системи, як правило, вказує на наявність підсистем та інших структурних компонентів і є багаторівневою ієрархічною схемою взаємодії підпрограм з управління.

Розробка структурної схемі є однією з найважливіших єтапів проектування програмного забезпечення, тобто описом сукупностей всіх компонентів, їх зв’язками, методами обміну та об’єктами між ними. Наявна структурна схема є певного роду мапою, та дозволить легко модифікувати або змінювати компоненти системи. Розроблена структурна схема відображена у додатку Б.

Структурна схема була поділена на логічні модулі, відокремлені джерела даних, презентація.

Презентація - інтерфейс користувача, це REST API для користувача, що інтегровано у зовнішню систему, та веб-інтерфейс для задач, пов’язаних з адмініструванням системи.

Джерело даних користувачів - це реляційна база даних що зберігає дані зовнішньої системи, доступ до якої є контрольованим, формат даних є гнучким, та налаштовується з зовнішньої системи. Контроль, та зміна статусу персональних даних відбувається за допомогою конфігураційного файлу.

Джерело даних відкладених задач - це не реляційна база даних, що забезпечує контроль на над чергою виконання, та зберігає розклад виконання періодичних задач.

Модуль відображення - відповідає за взаємозв’язок користувача та системи. Основною задачею є генерування відповіді, та компонентів веб інтерфейсу що будуть передані у веб інтерфейс користувача. Включає у себе сім розділів: перегляд аналітичних даних, зміна формату даних, видалення користувачів, перегляд користувачів, перегляд даних користувача, авторизація адміністратора, перегляд інформації щодо формату даних.

Модуль сервісів - відповідає за роботу бізнес логіки системі, надає контрольований доступ до джерела даних. Основною задачею є обробка запиту від

користувача або адміністратора, виконання операції з даними у джерелі даних, переадресація запиту до модуля відкладених задач, якщо задача потребує багато часу на виконання, для її виконання у фоновому режимі, формування даних для відповіді. Поділяється на п’ять головних сервісів: робота з аналітичними даними, робота з даними користувача, авторизація адміністратора, контроль відкладених задач, робота з джерелом даних.

Модуль відкладених задач - відповідає за роботу з довгими за часом виконання, та складними за кількістю задіяних ресурсів системи задачами. Головною метою є внесення змін до джерела даних користувача, контрольоване та періодичне копіювання джерела, відновлення джерела за потреби, з видаленням застарілих даних. Модуль містить власне джерело збереження даних, що відповідає за контроль виконання фонових задач, збереження розкладу задач, та контроль над процесами, що виконують задачі, та розподілення між ними навантаження. Основними сервісами є: робота з аналітичними даними, робота з користувачем, робота з бекапування даних.

Для виконання вимог, що були встановлені, було прийнято рішення щодо розробки архітектури системи максимально гнучкою. У розділі визначені загальні компоненти системи, та відокремлене представлення У архітектурі відокремлений сервіс для роботи з джерелом даних, що може бути за необхідності адаптований для роботи з будь-яким джерелом, без внесення глобальних змін до інших компонентів.

5 ВИБІР ТА ОБҐРУНТУВАННЯ ЕЛЕМЕНТІВ ТА ТЕХНОЛОГІЙ

Однією з найважливіших етапів розробки системи є вибір елементів та технологій. Перед початком розробки системи, згідно з розробленої схеми компонентів системи потрібно розглянути існуючі технології побудови, проаналізувати, та вибрати ті, що забезпечують найбільш гнучкий підхід до розробки, впровадження, та майбутнього масштабування системи. Згідно з обраних технологій потрібно встановити платформу розробки, мову програмування, бібліотеки, та фреймворки згідно з обраної мови програмування.

5.1 Особливості побудови складних систем

Взагалі, загальноприйнятого терміна "архітектура програмного забезпечення" не існує. На практиці, однак, потрібно усвідомлювати, який код хороший, а який - поганий. Перш за все, хороша архітектура - це вигідна архітектура, яка робить розробку та обслуговування програми простішою та ефективною. Програму з хорошою архітектурою простіше розширити, змінити, протестувати, налагодити та зрозуміти. Тобто ви фактично можете сформулювати перелік цілком розумних та універсальних критеріїв:

Ефективність системи. Перш за все, звичайно, програма має вирішувати завдання та виконувати свої функції добре та за різних умов. До них належать такі функції, як надійність, безпека, продуктивність, здатність керувати навантаженнями збільшується (масштабованість) тощо.

Гнучкість. Кожна програма повинна змінюватися з часом - змінюються вимоги, додаються нові. Чим швидше і зручніше можно вносити зміни до існуючих функцій, а також чим менше проблем і помилок буде - тим більш гнучкою і конкурентоспроможною буде система. Тому під час процесу розробки потрібно оцінити, що виходить із цього, і що, можливо, знадобитися змінити. Зміна одного фрагмента системи не повинна впливати на інші фрагменти. Там, де це можливо,

архітектурні рішення не повинні бути «вирізані з каменю», а вплив архітектурних помилок адекватно обмежений. Хороша архітектура дозволяє відмовлятися від ключових рішень і мінімізує помилок.

Розширюваність системи. Можливість додавати до системи нові об'єкти та функції, не порушуючи її основної структури. На початковій фазі має сенс включати в систему лише найосновніші та найважливіші функції. У той же час архітектура повинна забезпечувати легке розширення додаткових функцій.

Вимога щодо гнучкості та масштабованості архітектури системи (тобто здатності змінюватись і розвиватися) є настільки важливою, що вона навіть формулюється як окремий принцип - принцип відкритості та закритості - другий з п'яти принципів SOLID . Класи, модулі, функції, тощо - повинні бути відкритими для розширення, але не повинні бути закритими для зміни. Іншими словами, повинно бути можливим змінити поведінку системи без зміни існуючих частин системи.

Це означає, що програма повинна бути розроблена для зміни своєї поведінки та додавання нових функцій, написавши новий код (розширення) без зміни існуючого коду. У цьому випадку поява нових вимог не змінює існуючої логіки, але може бути реалізована головним чином через їх розширення. Цей принцип заснований на архітектурі плагінів.

Масштабованість процесу розробки. Можливість скоротити час розробки, додавши до проекту нових співробітників. Архітектура повинна допускати паралельний процес розробки, щоб багато людей могли працювати над програмою одночасно.

Тестованість. Простий код для тестування містить менше помилок і працює надійніше. Однак тести не тільки покращують якість коду. Багато розробників прийшли до висновку, що вимога "хорошої перевіряемості" - це також настанова, яка автоматично призводить до гарного дизайну і в той же час є одним з найважливіших критеріїв оцінки його якості. Навіть якщо не написано жодного рядка тестового коду, відповідь на це питання допоможе у 90% випадків зрозуміти, наскільки "хороший" чи "поганий" дизайн (ідеальна архітектура)

Перевикористання. Бажано розробити систему так, щоб її фрагменти могли бути повторно використані в інших системах. Добре структурований, читабельний і зрозумілий код.

Стійкість. Як правило, багато людей працюють над програмою - деякі ходять, приходять нові. Після написання супроводу програми, як правило, видається людям, які не були причетні до його розробки. Отже, хороша архітектура повинна зробити порівняно легким і швидким розуміння системи нових людей. Проект повинен бути добре структурований, не перетинатися не містити дублювання, мати добре оформлений код і бажано документацію. За можливістю в системі потрібно застосовувати стандартні, та загальноприйняті рішення що є звичними для подібних ситуацій. Чим екзотичніше система, тим складніше її зрозуміти іншим.

Незважаючи на безліч критеріїв, головне завдання при розробці великих систем - зменшити складність. Єдиним дієвим способом зменшення складності системи є її поділ на частини.

Поділ на частини - є ієрархічним розкладом. Складна система повинна складатися з невеликої кількості більш простих підсистем, кожна з яких складається з менших частин тощо, до тих пір, поки самі дрібні деталі не будуть досить легко зрозуміти та побудувати безпосередньо.

Важливим є те, що рішення зменшення складності за рахунок поділу є не тільки єдино відомим, проте й універсальним. Незважаючи на зниження складності, воно водночас забезпечує гнучкість, надає можливості для масштабування, а також дозволить підвищити стійкість системи за рахунок дублювання важливих частин.

Що стосується побудови архітектури програми, то це головним чином означає розбиття програми на підсистеми (функціональні модулі, сервіси, шари, підпрограми) та організацію їх взаємодії між собою та із зовнішнім світом. Чим незалежні підсистеми, тим безпечніше зосередитись на розробці кожної за одною, не турбуючись про всі інші частини.

У цьому випадку програма перетворюється на конструктор, що складається з набору модулів та підпрограм, які взаємодіють між собою за допомогою чітко

визначених та простих правил. Таким чином можно контролювати складність і скористатися всіма перевагами, які зазвичай пов'язані з концепцією "гарна архітектура":

- масштабованість - можливість розширення системи та збільшення її продуктивності за рахунок додавання нових модулів;
- ремонтопридатність - зміна або виправлення одного модулю без потреби оновлювати інші;
- замінність модулів - модуль можно легко змінити, або переключити на нову або стару версію;
- можливість тестування - модуль можно легко тестувати без потреби взаємодії у тесті з іншими модулями;
- перевикористання - можно перевикористати модуль при роботі з іншими.

Тобто, необхідність розділення складної проблеми на прості фрагменти - мета всіх методик проектування. У більшості випадків термін "архітектура" стосується лише результату такого поділу та конструктивних рішень, які практично не відрізняються від їх прийняття.

Систему розділяють на основні смислові блоки перед вибором об'єктів. У невеликих додатках часто достатньо двох рівнів ієрархії - система спочатку підрозділяється на підсистеми та пакети, а пакети діляться на класи. Розподіл на модулі і підсистеми найкраще робити виходячи з тих завдань, які вирішує система. Основне завдання розбивається на складові її підзадачі, які можуть вирішуватися та виконуватися незалежно один від одного. Кожен модуль повинен відповідати за вирішення підзадачі і виконувати відповідну їй функцію.

Модуль - це не довільний шматок коду, а окрема функціонально осмислена і закінчена програмна одиниця (підпрограма), яка забезпечує рішення деякої задачі і в ідеалі може працювати самостійно або в іншому оточенні і бути перевикористаною. Модуль повинен бути цілісним, та здатним до відносної самостійності в поведінці та розвитку.

Слід прагнути до того, щоб модулі були гранично автономні, це є ключовим параметром правильної декомпозиції. Тому проводити її потрібно таким чином, щоб модулі спочатку слабо залежали один від одного. Але крім того, є ряд спеціальних технік і шаблонів, що дозволяють потім додатково мінімізувати і послабити зв'язку між підсистемами. Необхідно розуміти, що мова йде про всіх підсистемах і послаблювати зв'язаність потрібно на всіх рівнях ієрархії, тобто не тільки між класами, але також і між модулями на кожному ієрархічному рівні.

Таким чином, грамотна декомпозиція ґрунтується, насамперед, на аналізі функцій системи і необхідних для виконання цих функцій даних.

5.2 Обґрунтування вибору технології для побудови серверної частини

Виходячи з критеріїв побудови архітектури системи, було обрано мовою програмування - гнучку, динамічно типізовану мову Python. Було протестовано найпопулярніші, та стабільні фреймфорки:

- Aiohttp - асинхроний фреймворк, що базується на Asyncio [11];
- Bottle - швидкий, простий та легкий WSGI мікрофреймворк;
- Django - веб-фреймворк що містить у собі велику кількість функцій з коробки;
- Falcon - високопродуктивний фреймворк для побудови хмарних API;
- Flask - мікрофреймворк що базується на Werkzeug та Jinja2;
- Muffin - асинхронный фреймворк базирующийся на Asyncio и Aiohttp;
- Pyramid - невеликий, швидкий та зрозумілий веб-фреймворк;
- Tornado - асинхронна мережева бібліотека і веб-фреймворк.

Тести проводилися на Amazon EC2. Всі додатки (крім Tornado) запускалися за допомогою Gunicorn (2 процесу на кожне). Для синхронних WSGI бібліотек використовувався Meinheld worker. Додаток на Tornado запускав 2 процеси, використовуючи засоби самого фреймворка. Кожна програма тестувалася за трьома основними сценаріями: закодувати невеликий об'єкт в JSON і повернути клієнту; завантажити відповідь від іншого сервера і повернути його клієнту; використовуючи

ORM завантажити колекцію об'єктів з бази, додати до неї ще один і відобразити список в шаблоні. За базу даних використовувався Postgresql з стандартними налаштуваннями.

Перший сценарій це своєрідний «Hello World!». Лише трохи ускладнений процесом кодування в JSON, що практично не впливає на результати. Результати було зображено на рисунку 5.1

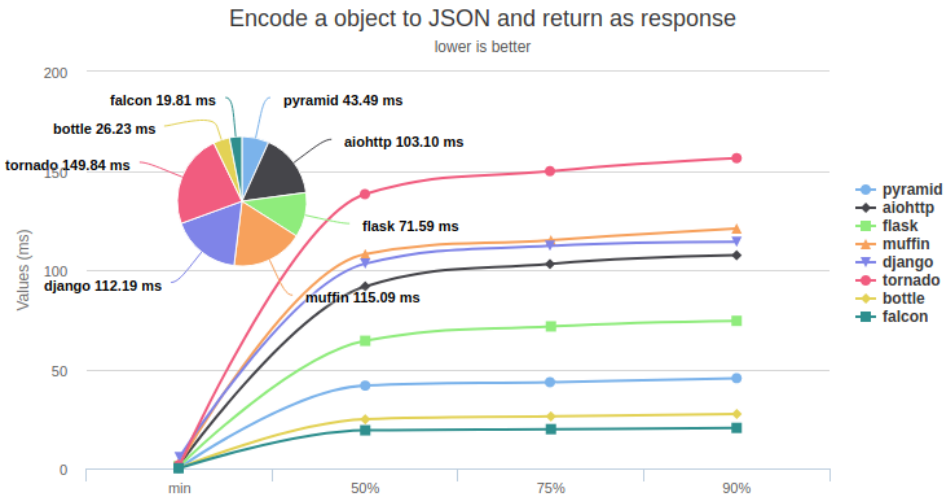

Рисунок 5.1 – JSON-тест фреймворків

У першому простому тесті з хорошим відривом перемогли синхронні фреймворки. Крім Django, але в виправдання останнього, можна сказати, що по-замовчуванню цей фреймворк робить безліч роботи. Асинхронні фреймворки ділять місця аутсайдерів і на останньому місці опинився Tornado. Дуже хороші результати у Falcon і Bottle.

Другий сценарій кілька синтетичний, так як його результати досить передбачувані. Тим не менш він повинен показувати наскільки добре платформа справляється з тривалими операціями очікування під час обробки запиту. Додатки зверталися до nginx налаштованому на відповідь із затримкою 100ms.

Резульати тесту зображені на рисунку 5.2.

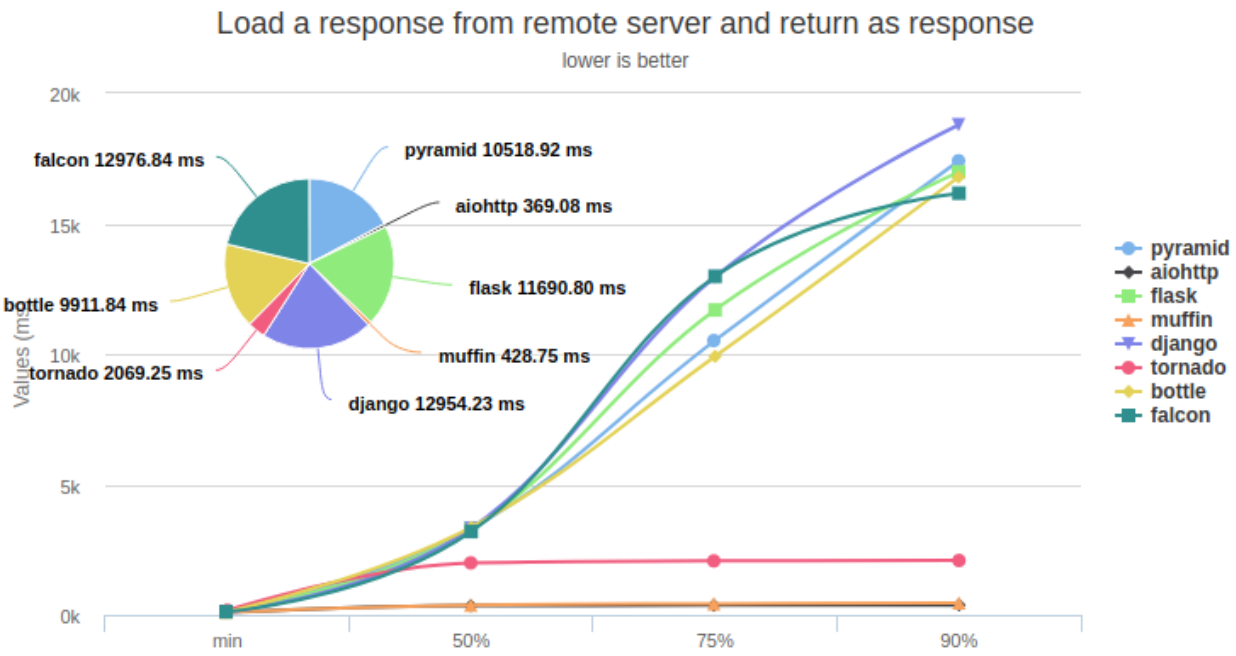

Рисунок 5.2 – Remote-тест фреймворків

Додатки зверталися до nginx налаштованому на відповідь із затримкою 100ms. Через це результати синхронних фреймворків дуже близькі, практично вся їх робота після певного моменту зводилася до очікування. Знову несподівані результати від Tornado, здавалося, що його заміри повинні були бути близькими до замірів Aiohttp і Muffin. Але тим не менш Tornado в 10 разів ефективніше в цьому тесті ніж найближчий синхронний фреймворк.

Третій сценарій це комплексний тест і імітує реальне життя, а саме використання бази даних, ORM, движка шаблонів.

Результати Aiohttp можна ігнорувати тк на жаль понад дві третини запитів повернули 502 помилки. На першому місці несподівано, виявився фреймворк Muffin досить швидко обробляє цей тест. Django значно програє, лише підтверджуючи повільність стандартного компонента обробки шаблонів і ORM.

Резульати тесту зображені на рисунку 5.3.

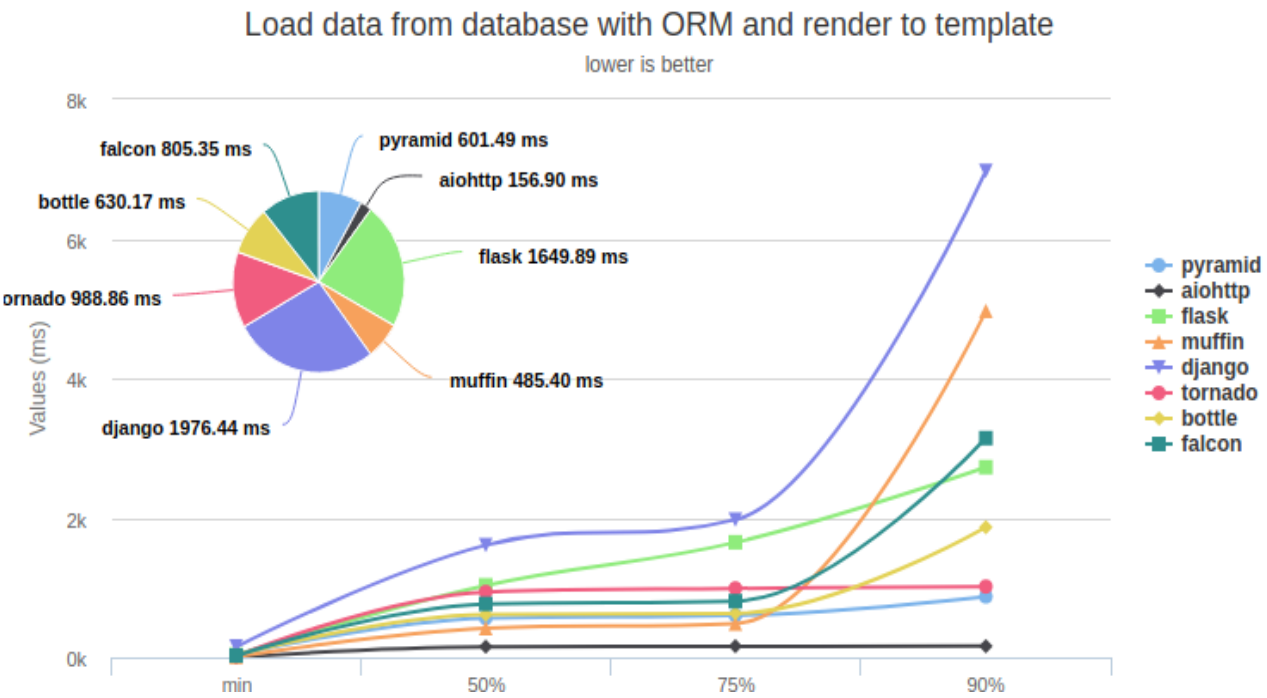

Рисунок 5.3 – Complete-тест фреймворків

Розглянувши фреймворки, та провівши тести, було обрано aiohttp технологією що буде допомогати у реалізації серверної частини додатку, тому що виходячи з тестів, цей фремворк набрав найбільшу кількість балів порівняно з іншими, а також найбільше підходить, виходячи з поставлених задач.

5.3 Обґрунтування вибору системи управління даними

Виходячи з архітектури системи, в майбутньому вона може бути адаптована до будь-якої системи управління даних, це є ключовим фактором, тому що система співпрацює з системою управління даних, що використовується у зовнішній системі.

Проте, для початку було розглянуто найпопулярніші СУБД, такі як MySQL, MsSQL, PostgreSQL. Вони є реляційними, що зберігає цілісність даних, і є дуже важливим фактором с точки зору надійності системи.

MySQL [12] - це безкоштовний пакет програм, однак нові версії виходять постійно, розширюючи функціонал і покращуючи безпеку. Існують спеціальні платні версії, призначені для комерційного використання. У безкоштовної версії найбільший

наголос робиться на швидкість і надійність, а не на повноту функціоналу, який може стати і гідністю і недоліком - в залежності від області застосування.

Переваги системи MySQL:

- розповсюджується безкоштовно;
- системно та цілісно документована;
- пропонує багато функцій, навіть у безкоштовній версії;
- пакет MySQL включений в стандартні репозиторії найбільш поширених дистрибутивів операційної системи Linux, що дозволяє встановлювати її елементарно;
- підтримує набір призначених для користувача інтерфейсів;
- може працювати з іншими базами даних, включаючи DB2 і Oracle.

Недоліки системи MySQL:

- недостатня надійність, у питаннях надійності деяких процесів по роботі з даними (наприклад, зв'язок, транзакції, аудит) MySQL поступається деяким іншим;
- як і багатьом іншим програмним продуктам з відкритим кодом, MySQL бракує деякого технічної досконалості, що часом позначається на ефективності процесів.

PostgreSQL є одним з декількох безкоштовних популярних варіантів систем управління даними. Це була одна з перших розроблених систем управління базами даних, тому в даний час вона добре розвинена, і дозволяє користувачам управляти як структурованими, так і неструктурованими даними. Може бути використана на більшості основних платформ, включаючи Linux. Прекрасно справляється з завданнями імпорту інформації з інших типів баз даних за допомогою власного інструментарію.

Ядро бази даних може бути розміщене в ряді середовищ, в тому числі віртуальних, фізичних і хмарних. Найсвіжіша версія пропонує обробку великих обсягів даних і збільшення числа одночасно працюючих користувачів. Безпека була покращена завдяки підтримці DBMS_SESSION.

Переваги PostgreSQL [13]:

- є масштабованим і здатний обробляти терабайти даних;

- підтримує формат json;
- існує безліч визначених функцій;
- доступний ряд інтерфейсів.

Недоліки PostgreSQL:

- складне початкове конфігурування;
- швидкість роботи під час пакетних операцій.

Microsoft SQL-сервер - це система управління базами даних, ядро якої працює на хмарних серверах, а також локальних серверах, причому можна комбінувати типи застосовуваних серверів одночасно. Незабаром після випуску Microsoft SQL сервер 2016, Microsoft адаптувала продукт для операційної системи Linux, а на Windows-платформі він працював спочатку.

Переваги MsSQL:

- продукт дуже простий у використанні;
- поточна версія працює швидко і стабільно;
- ядро надає можливість регулювати рівні використання ресурсів.

Недоліки MsSQL:

- ціна виявляється дуже великою порівняно з іншими СУБД;
- навіть при ретельній налаштування продуктивності корпорація SQL Server здатний зайняти всі доступні ресурси;
- повідомляється про проблеми з використанням служби для імпорту файлів.

Розглянувши системи управління даними, було вирішено використовувати PostgreSQL, як найгнучку, безкоштовну та придатну для роботи з великою кількістю даних СУБД.

5.4 Обґрунтування вибору технології для реалізації відкладених задач

Асинхронне або неблокуюче виконання операцій [14] - це такий метод виконання, при якому деякі завдання виконуються окремо від основного потоку

програми. Такий підхід дає кілька переваг, одне з яких - безперервна робота коду на стороні користувача.

Передача повідомлень - це метод, за допомогою якого компоненти програми можуть взаємодіяти один з одним і передавати інформацію. Вона може бути виконана як синхронно, так і асинхронно, а так само вона забезпечує процес комунікації між окремими частинами програми. Передача повідомлень часто застосовується в якості альтернативи баз даних, причому такий метод забезпечує додатковий функціонал як поліпшення продуктивності і робота в оперативній пам'яті.

Celery - черга завдань, побудована на системі асинхронної передачі повідомлень. У програмуванні, Celery можна використовувати в якості сховища для відкладених завдань. Програма, яка передала завдання, може безперешкодно продовжувати працювати, а згодом вона звертається до celery для підтвердження закінчення процесу обчислення, щоб отримати необхідні дані.

Незважаючи на те, що celery написана на Python, працювати з нею можна при використанні будь-якої мови програмування за допомогою webhooks.

Отже Celery знижує навантаження на продуктивність, виконуючи частину функціональності у вигляді відкладених завдань або на тому ж сервері, що і інші завдання, або на іншому сервері.

5.5 Обґрунтування вибору технології для реалізації інтерфейсу користувача

Сучасним методом розробки інтерфейсу користувача є розробка за допомогою принципу односторінкового додатку.

Односторінкові додатки працюють в рамках браузера і не вимагають перезавантаження сторінки або завантаження додаткових сторінок під час використання

Односторінкові додатки створені спеціально для того, щоб надавати користувачам відмінний UX, що нагадує «природну» середовище браузера без перезавантажень сторінок, а значить, без затримок при здійсненні дій.

Типове односторінковий додаток виглядає як веб-сторінка, що довантажує і оновлює контент без перезавантаження за допомогою JavaScript. Клієнт запитує розмітку сторінки і її контент, а потім створює кінцевий вигляд сторінки безпосередньо в браузері. Було розглянуто та досліджено три найбільш популярні фреймворки для побудови односторінкового додатку.

Angular вперше був випущений в 2010 році, розроблений Google. Він заснований на TypeScript та є JavaScript-фреймворк. У Angular компоненти називаються директивами, які використовуються в якості маркерів для елементів об'єктної моделі документа (DOM), вони можуть відслідковувати і визначати конкретну поведінку кожного окремого компонента.

React - був випущений в 2013 році Facebook. Крім Facebook, він використовується Instagram і WhatsApp. React зазвичай об'єднує призначений для користувача інтерфейс і поведінку компонентів. Наприклад, код, який може створити в React компонент «hello world», і та ж частина коду, відповідає за розробку елемента користувальницького інтерфейсу, а також відстеження його поведінки. У React поновлення між версіями більш зручні, ніж в Angular і Vue. Більш того, такі скрипти, як react codemod, допоможуть мігрувати. Проти React не є повноцінним фреймворком, і для розширених функцій йому потрібні сторонні бібліотеки, що робить його модульним, проте не забезпечує цілісного функціоналу.

Vue.js [16] - наймолодший з усіх трьох. Архітектуру компонентів Vue зображено на рисунку 5.4.

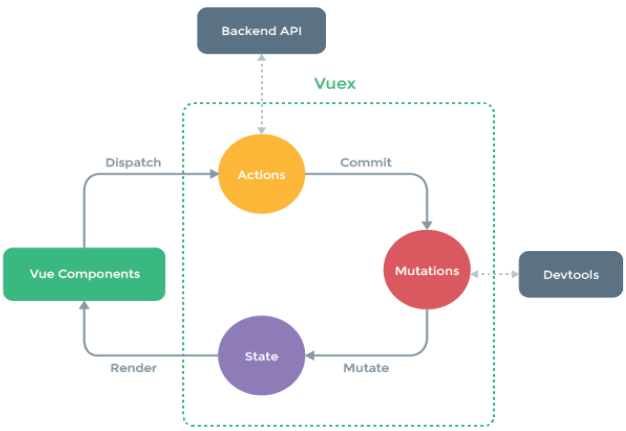

Рисунок 5.4 – Архітектурна схема Vue [16]

Він був розроблений колишнім співробітником Google в 2014 році. Vue надає можливість настройки, яка дозволяє комбінувати призначений для користувача інтерфейс і поведінку компонентів в одному скрипті. Крім того, він дозволяє використовувати препроцесори, а не CSS, що в даний час полегшує роботу розробників. Vue може інтегрувати інші бібліотеки, такі як Bootstrap. Дозволяє створювати функціональні додатки, які відповідають усім сучасним стандартам, при мінімальному підключенні нових ресурсів. Завдяки використанню будь-яких шаблонів і доступності документації, більшість виникаючих проблем вирішуються досить швидко. У тому числі в порівнянні з React, так як в більшості додатків, які не мають складних інтерфейсів, вся міць цього фреймворка є трохи надлишковою.

Розглянувши фреймворки для побудови інтерфейсу користувача, було обрано Vue, як найпотужніше, та найоптимальніше рішення для виконання поставлених задач. Цей фреймворк є мінімалістичним, швидким, та сучасним, має високу здатність для масштабування.

Vue більш гнучкий ніж Angular, і підтримує безліч різних систем збірок, не обмежуючи розробників в тому, яку структуру використовувати для додатка.

5.6 Висновки до розділу

У цьому розділі було розглянуто та визначено основні технології системи, та розкрите питання побудови архітектури складної системи.

Для побудови серверної частини системи було обрано мову програмування Python та асинхронний фреймворк aiohttp, що був протестованим, та обраним, як найшвидший, та той який має у собі весь необхідний інструментарій, для інтеграції з зовнішньою системою та виконання операцій зі сховищем даних.

Модуль для виконання відкладених задач використає бібліотеку Celery, що написана під мову програмування Python, проте може бути інтегрована та використана за допомогою будь-якої мови програмування.

Для розробки інтерфейсу користувача було обрано технологію односторінкових додатків, що забезпечує найбільшу продуктивність, та модульність клієнтської частини системи. Фреймворком для розробки інтерфейсу було обрано Vue.js, тому що він є мінімалістичним, та включає у себе весь необхідний інструментарій, для побудови клієнтської частини.

Системою збереження даних було обрано PostgreSQL, тому що вона здатна обробляти велику кількість даних, та є надійною з точки зору безпеки, що стало ключовим фактором при виборі.

6 РЕАЛІЗАЦІЯ БІЗНЕС-ЛОГІКИ СИСТЕМИ

Обравши технології побудови системи, та проаналізувавши вимоги до системи, було визначено, що система повинна бути модульна, розподілена на сервіси. Для реалізації бізнес-логіки системи потрібно обрати шаблони проектування, за якими буде вестись розробка, провести основну частину побудови бізнес-логіки, налаштувати веб-сервер для розгортання інтерфейсу користувача, та інтерфейса системи побудованого на базі REST, та контейнерізовати систему для майбутнього масштабування.

6.1 Узагальнення шаблонів проектування

Шаблон проектування або модель в розробці програмного забезпечення - це повторювана архітектурна конструкція, яка забезпечує вирішення проблеми дизайну в загальному контексті.

Взагалі шаблон не є повним прикладом, який можна перетворити безпосередньо в код. Це лише один приклад вирішення проблеми, яка може бути використана в різних ситуаціях. Об'єктно-орієнтовані шаблони показують взаємозв'язки та взаємодії між класами чи об’єктами, не визначаючи, які кінцеві класи чи об’єкти програми використовуються.

Шаблони мають ряд переваг перед самостійним проектуванням. Основна перевага використання шаблонів - зменшення складності розробки готовими абстракціями для вирішення цілого класу проблем. Шаблон дає рішенню свою назву, що полегшує спілкування між розробниками та дозволяє звертатися до знайомих шаблонів. Шаблони допомагають уніфікувати деталі рішень: модулі, елементи проекту - кількість помилок зменшується.

Використання шаблонів концептуально схоже на використання готових бібліотек коду. Правильно сформульована модель дизайну дозволить знайти вдале рішення і використовувати його знову і знову.

Для побудови системи було за основу було взято шаблон проектування веб-застосунків - MVC [17].

MVC - це шаблон проектування веб-додатків, який містить кілька менших шаблонів. При використанні MVC користувальницький інтерфейс,модель даних, інтерфейс користувача поділяються на три окремі компоненти, тому зміна одного компонента має мінімальний або ніякого впливу на інший.

Основна мета шаблону - відокремити дані та бізнес-логіку від візуалізації (зовнішнього вигляду). Цей підрозділ розширює можливість повторного використання коду та спрощує технічне обслуговування (наприклад, зміни зовнішності не відображаються в бізнес-логіці).

Концепція MVC розділяє дані, перегляди та обробку дій користувача на компоненти (модель, перегляд, контролер).

Модель - це об'єктна модель будь-якої теми, містить дані та методи роботи з цими даними, відповідає на запити контролера, повертає дані та / або змінює свій статус. Модель не містить інформації про те, як дані можна візуалізувати і не «спілкується» безпосередньо з користувачем.

Перегляд - відповідає за відображення інформації (візуалізацію), одні і ті ж дані можуть відображатися різними способами.

Контролер - забезпечує зв’язок між користувачем і системою, та використовує модель та зовнішній вигляд для реалізації необхідної реакції на дії користувача, як правило, на рівні контролера.

Для побудови роботи з джерелом даних, виходячи з того, що система повинна адаптуватися до будь-якої системи управління базами даних, було обрано породжуючий шаблон проектування - абстрактна фабрика.

Абстрактна фабрика - це породжуючий шаблон проектування, який дозволяє створювати сімейства пов'язаних об'єктів, не прив'язуючись до конкретних класів створюваних об'єктів.

Шаблон надає інтерфейс для створення сімейств взаємопов'язаних об'єктів з певними інтерфейсами без вказівки конкретних типів даних об'єктів.

Відповідно до шаблону проектування, система не повинна залежати від того, як об’єкти створюються, компонуються та відображаються. Сімейство взаємопов'язаних об'єктів потрібно використовувати разом, і слід забезпечити дотримання цього обмеження. Необхідно забезпечити бібліотеку об'єктів, яка містить лише її інтерфейси, а не реалізацію.

Також, для роботи з джерелами даних, потрібно зберігати та передавати у системі єдиний відкритий зв’язок з базою, це можно реалізувати за допомогою шаблону Singleton.

Singleton - створюючий шаблон проектування, який гарантує, що в однопоточному застосунку буде лише один екземпляр класу і забезпечує глобальну точку доступу для цього екземпляра.

Клас має лише один екземпляр і забезпечує глобальну точку доступу. Якщо спробувати створити цей об’єкт, він буде створений, лише якщо він ще не існує. В іншому випадку повертається посилання на існуючий екземпляр і не відбувається нового розподілу пам'яті.

Було обрано загальні шаблони проектування, що використані при побудові системи. Загальними перевагами цих шаблонів є:

- перевірені рішення - при розробці системи використовується менше часу, використовуючи готові рішення, замість повторного винаходу велосипеда. До деяких рішень ви змогли б додуматися і самі, але багато хто може бути для вас відкриттям;
- стандартизація коду - при розробці системи вникає меньше прорахунків при проектуванні, використовуючи типові уніфіковані рішення, так як всі приховані проблеми в них вже давно знайдені;

6.2 Побудова веб сервера на основі aiohttp

Aiohttp - бібліотека, що надає можливість асинхроно опрацьовувати вхідні HTTP запити, та надавати відповідь на них. Обробник запиту - підпрограма яка приймає екземпляр Request як єдиний параметр і повертає екземпляр Response.

Щоб створити сервер, необхідно створити екземпляр класу що розташований за шляхо aiohttp.web.Application, та виступаю корневим об'єктом.

При створенні об'єкта у якості параметра передається, безкінечний цикл, що зберігає потоки виконання, та розподіляє навантаження між ними.

Безкінечний цикл, в якому обертаються обробники, є основою для aiohttp. Обробник - це так звана коренова програма - об'єкт, який не блокує введення та виведення. Даний тип об'єктів з'явився в python 3.4 у бібліотеці asyncio. Поки всі обчислення не виконані на цьому об'єкті, він наче «засинає», тоді як перекладач може обробляти інші об’єкти.

Маршрутизація системою виконується за допомогою об'єкту Router, що являє собою двовимірний масив, з елементами. Перший елемент - метод http, URL йде далі, обробник - третій у ланцюжку, а назва маршруту - остання. Потім список маршрутів імпортується в app.py, і вони заповнюються простим циклом для програми.

Middlewares - обробники що виконуються перед попаданням запиту до функції відображення, що реалізовано за допомогою шаблону проектування - фабрика. Вони викликаються почергово, приймають запит, змінюють його, та передають далі на виконання. Ці обробники конфігуруються у екземплярі корневого класу.

Відображення - функція, що приймає запит, та повертає у відповідь екземпляр класу Response, виключення або перенаправляє на інше посилання.

Загальний алгоритм обробки запиту відображено на рисунку 6.1.

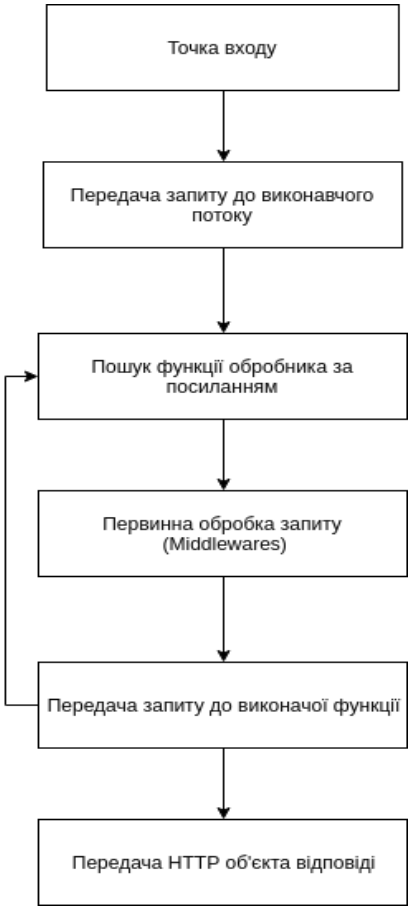

Рисунок 6.1 – Алгоритм обробки запиту на aiohttp.

Для функціонування додатку було створено наступні обробники запитів:

- авторизація користувача (отримання ключу доступу);
- вихід з облікового запису (видалення ключу);
- перегляд аналітичної інформації;
- отримання списку користувачів;
- запит на видалення користувача (блок схема роботи у додатку Д);
- запит на створення резервної копії (блок схема роботи у додатку Е);
- отримання даних користувача (схема переходів у додатку Ж);
- отримання даних щодо формату збереження даних;
- зміна формату збереження даних.

Для авторизації у системі було створено два обробники.

Перший - авторизація за допомогою створення секретного ключа доступа (для адміністратора), алгоритм авторизації зображений на рисунку 6.2.

Рисунок 6.2 – Алгоритм авторизації адміністратора.

Було розроблено функцію-обробник, що приймає дані адміністратора (які саме - вказано у конфігураційному файлі), шукає на основі них запис у базі за співпадінням параметрів, та створює запис у таблиці з ключами з посиланням на нього та ключем доступу у базі даних. Створений ключ повертається користувачу у відповідь.

Другий - авторизація за допомогою зовнішньої системи(для користувача). Було створено обробник, що приймає секретний ключ зовнішньої системи, та ідентифікатор користувача, тим самим авторизує його у системі.

6.3 Сервіс для роботи з джерелом даних

Виходячи з потреб системи, сервіс повинен адаптуватися до будь-якого джерела даних, з будь-якою структурою. Сервіс повинен автоматично аналізувати

структуру даних, шукати зв’язки між таблицями, а також створювати оптимізовані запити.

Було прийнято рішення про створення власної ORM [18] технології для роботи з базами даних за принципом роботи “table first” - аналізуючи, та приводячи структуру бази до екземплярів класів на мові програмування Python.

ORM - це технологія програмування, яка поєднує бази даних з поняттями об'єктно-орієнтованих мов програмування для створення "віртуальної бази даних".

Проблематика системи полягає у тому, що вона не знає, у якому саме форматі зовнішня система зберігає дані, а лише орієнтується на конфігураційний файл, у котрому повідомлено про те, яка таблиця є ключовою, та від якої таблиці потрібно починати побудову структури, цю проблему можно вирішити за допомогою віртуалізації структури бази у вигляді об'єктів.

Даний підхід дозволяє абстрагуватися від конкретної системи управління базами даних, та, використовуючи шаблон проектування - абстрактна фабрика, побудувати загальний інтерфейс доступу до даних.

Основними задачами для ORM є:

- зчитування структури бази, та її побудови у вигляді коду;
- запит на видалення даних, виходячи з структури;
- зчитування даних, виходячи з структури;
- створення запису з ключем авторизації, спираючись на конфігурацію.

Для роботи системи було реалізовано загальний інтерфейс, для роботи з адаптерами до бази даних, та реалізований адаптер до системи управління базою даних PostgreSQL. Також було розроблено скрипти для створення резервних копій даних.

6.4 Сервіс відкладеного виконання

Для реалізації сервісу відкладеного виконання, за основу було взято систему контролю за чергами задач Celery.

Celery [19] - це проста, гнучка та надійна розподілена система для обробки величезної кількості повідомлень, забезпечуючі операції з інструментами, необхідними для підтримки такої системи, тобто черга завдань з акцентом на обробку в режимі реального часу, а також підтримує планування завдань.

Черги завдань використовуються як механізм розподілу роботи по потокам або машинам. Вхід черги завдань - це одиниця роботи, яка називається завданням. Виділені робочі процеси постійно відстежують черги завдань, щоб виконувати нові роботи.

Celery спілкується за допомогою повідомлень, зазвичай за допомогою посередника для посередництва між клієнтами та працівниками. Щоб ініціювати завдання, клієнт додає повідомлення до черги, тоді брокер доставляє це повідомлення працівникові. Може складатися з декількох працівників і посередників, що дозволяє підвищити її продуктивність за допомогою горизонтального масштабування.

Основними функціями Celery є:

- виконувати завдання асинхронно або синхронно;
- виконувати періодичні завдання;
- виконувати відкладені завдання;
- розподілене виконання, може бути запущена на декількох серверах;
- в межах одного процесу - можливе конкурентну виконання декількох завдань;
- виконувати завдання повторно, якщо було отримано помилку;
- обмежувати кількість завдань в одиницю часу;
- моніторити виконання завдань;
- виконувати підзавдання;
- надсилати звіти про помилки на пошту;
- перевіряти чи виконалось завдання.

Схема роботи Celery зображена на рисунку 6.3.

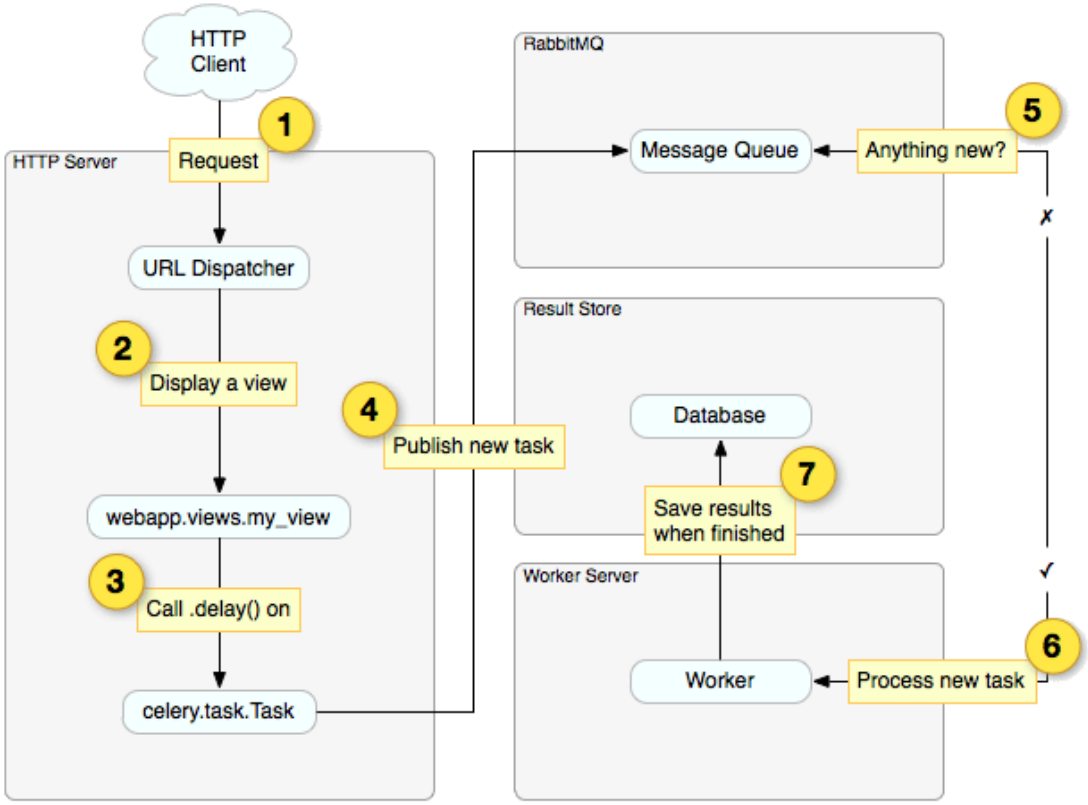

Рисунок 6.3 – Схема роботи Celery. [19]

Схема конкурентності задач зображена у додатку В.

Для відстежування виконання задач, та їх адміністрування було використано веб застосунок - Flower [20]. Приклад інтерфейса застосунку зображено на рисунку 6.4.

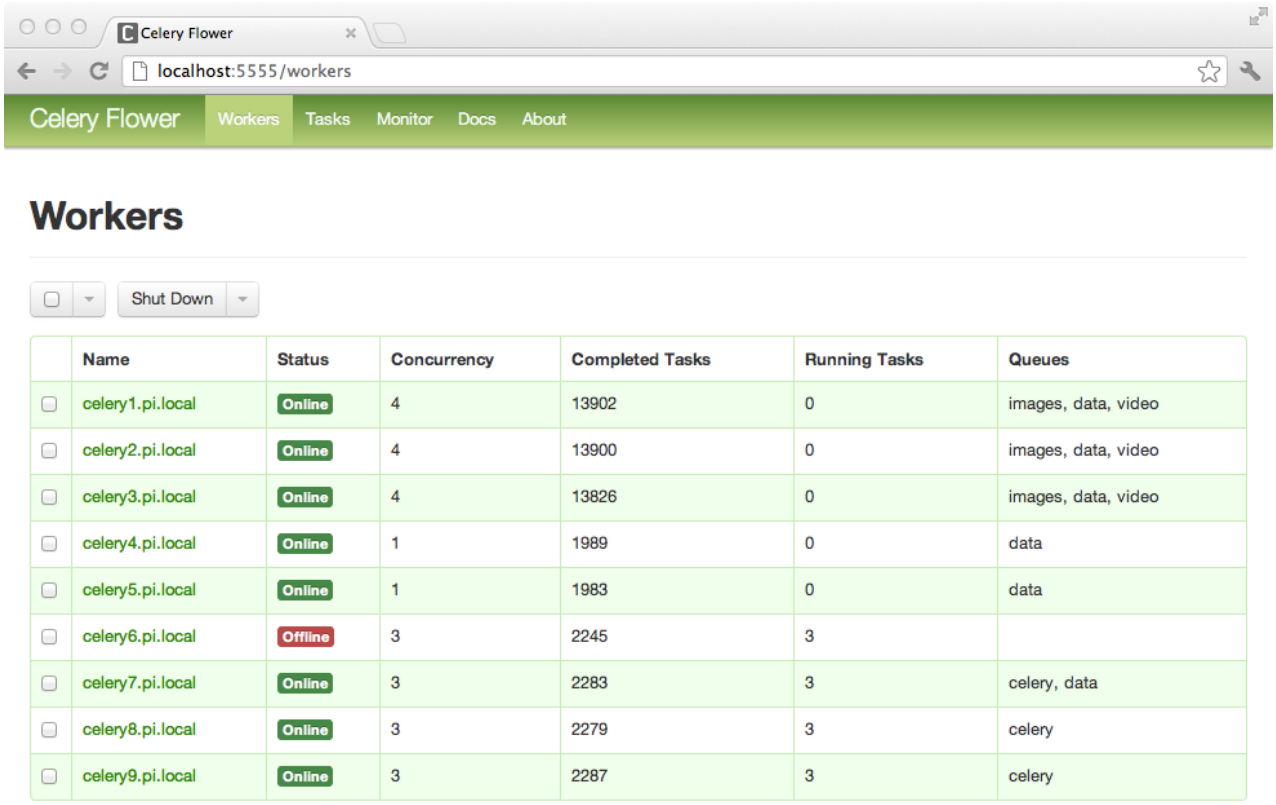

Рисунок 6.4 – Інтерфейс веб застосунку Flower. [20]

Flower має ряд переваг, таких як:

- моніторинг задач у реальному часі;
- можливість показу деталей задач, таких як аргументи;
- перегляд статусу виконавчих процесів;
- перезапуск процесів;
- відображення графіків, щодо статусу виконаних задач.

Використовуючи інструментарій Celery, було розроблено сервіс, що містить функції роботи з даними:

- відновлення резервних копій, з подальшим видаленням застарілих даних;
- формування даних користувача для видачі;
- видалення застарілих даних.

6.5 Конфігурація nginx

Для розгортання системи було обрано HTTP веб-сервер Nginx.

Nginx - це веб-сервер під управлінням операційних систем Unix. Виробник позиціонує його як простий, швидкий і надійний сервер, не перевантажений функціями.

Під час конфігурації сервери Nginx поділяються на віртуальні сервери. Віртуальні сервери діляться на місця. Для віртуального сервера ви можете вказати адреси та порти, які будуть приймати з'єднання, а також імена, що можуть містити "*", щоб вказати будь-який порядок у першій та останній частині, або бути визначеними регулярним виразом.

Було визначено два віртуальних сервера для роботи графічного інтерфейсу, та для роботи з REST інтерфейсом, що знаходиться за шляхом “/api/”. Також було сконфігуровано логування запитів у файли.

Серверна частина додатку додатку була сконфігурована за допомогою WSGI, стандартом взаємодії між Python-програмою, що виконується на стороні сервера, і самим веб-сервером.

6.6 Система моніторингу Sentry

Під час роботи системи, важливо відстежувати помилки, що виникають у роботі з нею. Найбільш зручним інструментом для мониторингу помилок, з зручним інтерфейсом користувача - є система Sentry [21].

Sentry є додатком, що вбудовується в систему для оперативного моніторингу помилок. Інтерфейс являє собою веб-застосунок зі списком помилок і можливістю виконувати над ними різні дії, приклад інтерфейсу зображений на рисунку 6.5.

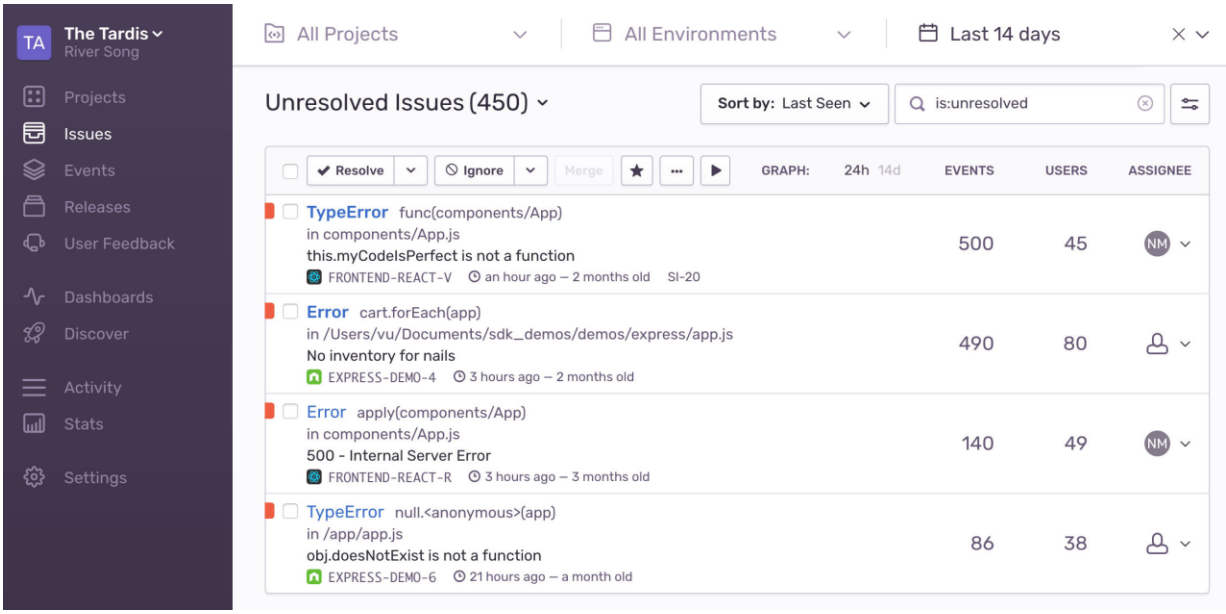

Рисунок 6.5 – Веб-інтерфейс системи моніторингу sentry [21]

Основні можливості:

- список помилок оновлюється в режимі реального часу;
- якщо помилка була позначена як вирішена і з'явилася знову, то вона знову створюється і враховується в окремому потоці;
- помилки групуються і відображаються в порядку частоти появи;
- помилки можна фільтрувати по статусах, джерелам та рівням логування, імені сервера і т.д.

Параметри задаються як частина ініціалізації клієнта. Клієнт - це клас, який може бути екземпляром певної конфігурації, і вся звітність може відбуватися з екземпляра цього об'єкта. Зазвичай екземпляр створюється десь у всьому світі, а потім імпортується за необхідності. Для початку все, що потрібно, це DSN.

DSN - це посилання на проект що розташований у системі моніторингу, і використовується для ідентифікації сервіса. Містить у собі унікальний відкритий ключ сервіса.

Для інтеграції потрібно встановити дві залежності:

- aiocontextvars;
- sentry-sdk.

Налаштування відбувається виклику методу sentry_sdk.init у який буде передано DSN та екземпляр обробника AioHttpIntegration у якості аргументів функції. У результаті було додано систему моніторингу, та інтегровано її до загальної системи.

6.7 Контейнеризація сервісів

Для легкого підняття сервісів, гнучкого масштабування та балансування навантаження було обрано систему контейнеризації Docker.

Docker - це програмне забезпечення, яке автоматизує розгортання та управління додатками в контейнерних середовищах. Дозволяє "додавати" додаток із усіма його середовищами та залежностями в контейнер, котрий можна буде перенести в будь-яку систему Linux з підтримкою групи управління ядром, а також забезпечує середовище управління контейнерами, також обмежує контейнери, змушуючи їх працювати, як єдиний процес.

У однопроцесних контейнерів багато переваг, включаючи прості і більш дрібні поновлення. Не потрібно зупиняти процес баз даних, коли потрібно оновити тільки веб-сервер, також наявна ефективна архітектура для того, щоб будувати додатки, засновані на однопроцесних.

У однопроцесних контейнерів також є обмеження. Не можна запускати агенти, скрипти реєстрації або автоматично виконувати SSH-процеси всередині контейнера. Також нелегко незначно оновлювати контейнер на рівні додатку. Доведеться запускати новий оновлений контейнер.

Контейнери Docker зроблені так, щоб бути більш безструктурними, ніж Linux Containers.

По-перше, Docker взагалі не підтримує зовнішнє сховище. Він обходить це тим, що дозволяє системі підключати сховище хоста в якості томів Docker з контейнерів. Так як томи підключаються, вони не вважаються частиною середовища контейнера.

По-друге, контейнери Docker складаються з шарів в режимі читання. Це означає, що як тільки створюється образ контейнера, він не змінюється. Під час виконання програми, якщо процес в контейнері змінює свій внутрішній стан, створюється різниця між внутрішнім станом і способом, з якого був створений контейнер.

Томи - це фізичні області дискового простору, що розділяється між хостом і контейнером, або навіть між контейнерами. Іншими словами, том - це спільний каталог у хості, видимий з деяких або всіх контейнерів.

Найважливіша перевага Docker - загальне відокремлення мережевих ресурсів, сховища і деталей ОС, у порівнянні з Linux Container. З Docker додаток дійсно не залежить від налаштувань цих низькорівневих ресурсів. При переміщенні контейнера від одного хоста Docker до іншої машини, він гарантує, що навколишнє середовище для додатка залишиться незмінною.

Docker застосовується для управління окремими контейнерами (сервісами), з яких складається програма, проте розгортання системи, у якості цілісного ядра, та відокремлюючи для неї власну мережу, було використано інструмент Docker Compose.

Docker Compose - це інструментальний засіб, що входить до складу Docker. Він призначений для вирішення завдань, пов'язаних з розгортанням проектів, використовується для одночасного управління декількома контейнерами, що входять до складу системи. Цей інструмент пропонує ті ж можливості, що і Docker, але дозволяє працювати з більш складними додатками.

Було сконфігуровано наступні контейнери:

- front-end - для розгортання веб інтерфейсу;
- celery - для розгортання сервісу відкладених задач;
- flower - для розгортання сервісу моніторингу за відкладеними задачами;
- web - серверна частина системи;
- sentry - моніторинг роботи системи;
- redis - брокер для сервісу відкладених задач;

- nginx - веб-сервер, для маршрутизації вхідних запитів;
- postgresql - система управління базою даних.

Схема архітектури та взаємодії докер контейнерів знаходиться у додатку Г.

6.8 Висновки до розділу

У даному розділі було описано, та за результатами опису, розроблено бізнес логіку системи. Вона є модульною, поділеною на сервіси.

Всі модулі системи було контейнерізовано за допомогою технологій Docker, та Docker-Compose, що дозволить у майбутньому проводити масштабування та гарячу заміну сервісів, а також покращить відказостійкість, завдяки автоматичного балансування навантаження. Серед переваг даного підходу є й підхід до розгортання середовища - “Clean code”, це означає що система здатна бути розгорнута на будь-якому сервері, з встановленою Unix подібною системою.

Розроблено сервіс авторизації, що надають безпечний, та контролюючий доступ до даних.

Також було сконфігуровано веб-сервер Nginx, для маршрутизації запитів до системи. Серед основних переваг веб-серверу можно виділити здатність автоматично забороняти спам доступ до системи, знеможливлюючи виникнення відказу системи через Ddos атаки. Також, перевагою є те, шо веб-сервер логічно розподіляє сервіси на віртуальні хости, тим самим дозволяючи розподіляти навантаження, та доступ до сервісів. Було налаштовано вбудовану систему логування запитів, для подальшого аналізу та виявлення помилок у разі необхідності.

Було розроблено сервіс що адаптується до будь-якої структури бази за мінімальною конфігурацією, котру можно встановити за допомогою інтерфейса адміністратора без необхідності внесення змін до кодової бази проекту.

Для швидкого реагування на помилки, та для їх зручного дослідження було інтегровано систему моніторингу Sentry. Важливим є те що інтегрована система

моніторингу вказує не лише на помилку, а на стрічку подій за якої вона виникла, представляючи зручний інтерфейс для відстеження.

Усі сервіси було розроблено з використанням загальноприйнятих шаблонів проектування, та принципів розробки складних систем.

7 РОЗРОБКА КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ

Для розробки клієнтської частини було обрано технологію односторінкового додатку, та бібліотеку Vue.js. В ядрі Vue.js знаходится система, яка дозволяє декларативно відображати дані в DOM за допомогою простих шаблонів, тобто дані і DOM реактивно пов'язані. Реактивність - це процес, коли при зміні значень змінної, буде виконана перерисовка всіх пов'язаних з нею елементів компонента, а також перераховані всі розрахункові властивості.

Функции Vue.js:

- реактивні інтерфейси;
- декларативний рендеринг;
- зв'язування даних;
- директиви;
- логіка шаблонів;
- компоненти;
- обробка подій;
- властивості;
- переходи і анімація CSS;
- фільтри.

Vue підходить для великих односторінкових додатків завдяки своїм основним компонентам, таким як Router і Vuex. З Vue можна як використовувати загальнодоступні API для створення додатків, так і реалізовувати виконуються сервером додатка. Але Vue найкраще підходить для розробки рішень, які використовують зовнішні API для обробки даних.

Vue добре підходять для обробки німих компонентів - невеликих, які не мають стану функцій, які отримують вхідні дані і повертають елементи в якості висновку.

7.1 Маршрутизація за допомогою Vue Router

Vue Router — офіційна бібліотека маршрутизації для Vue.js. Вона глибоко інтегрується з Vue.js і дозволяє легко створювати SPA-додатки.

Основними можливостями є:

- вкладені маршрути та уявлення;
- модульна конфігурація маршрутизатора;
- доступ до параметрів маршруту, query, wildcards;
- анімація переходів уявлень на основі Vue.js;
- зручний контроль навігації;
- автоматичне проставлення активного CSS класів для посилань.

Динамічне відображення відбувається за допомогою VueRouter, виконується завдяки компоненту router-view. По замовчуванню автоматично вбудовується активний компонент. Роутер має наслідкову деревоподібну структуру. Компонентами вищого порядку є:

- / - домашня сторінка системи, містить інформацію про систему;
- /auth/ - компоненти для авторизації та виходу з системи;
- /user/ - компоненти для роботи з користувачем;
- /system/ - компоненти для конфігурації та перегляду формату даних;
- /analytic/ - компоненти, що відображають загальну інформацію про стан системи. Повна схема маршрутизації зображена на додатку И.

7.2 Набір готових компонентів Vuetify

Vuetify - бібліотека Vue UI, що містить у собі набір готових базових компонентів, розроблених у ситі Material design.

Material design - візуальна мова, що синтезує класичні принципи гарного дизайну з інноваціями технології та науки.

Бібліотеку було встановлено за допомогою внутрішнього інтерфейсу командної строки, за виконанням команди: vue add vuetify. Після виконання, Vue завантажить та сконфігурує необхідні залежності, і компоненти будуть доступні для використання.

Було розроблено інтерфейс користувача, з використанням цієї бібліотеки, на рисунку 7.1 зображено форму авторизації у системі, що знаходиться за посиланням /auth/login/:

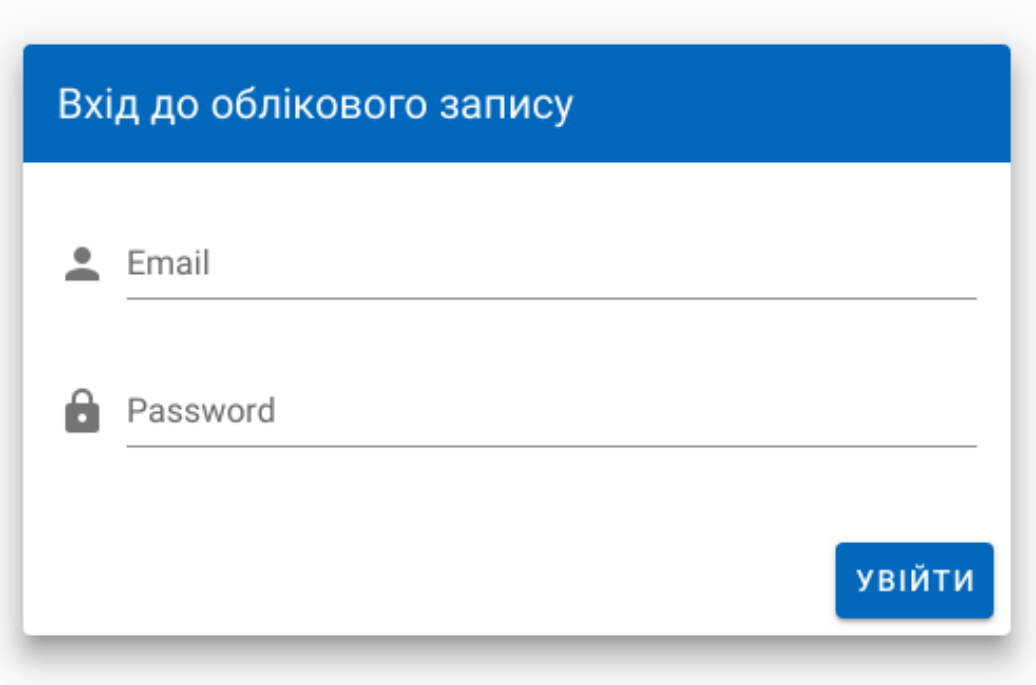

Рисунок 7.1 – Форма авторизації, розроблена за допомогою компонентів Vuetify

Vuetify - має велику кількість базових компонентів, що розроблені для максимізації зручності використання. Усі компоненти є стандартизованими, та семантично зрозумілими для користувача.

7.3 Висновки до розділу

Vue - є потужним засобом для розробки інтерфейсу користувача. Основним принципом роботи фреймворку є технологія односторінкового додатку, шо дозволяє динамічно оновлювати елементи веб-сторінки, без необхідності її повного оновлення.

У даному розділі було спроектовано та розроблено функціонал клієнтської частини, та побудовано графічний інтерфейс. Для реалізації маршрутизації між сторінками було використано Vue Router. Було розроблено структуру переходів між

компонентами, визначено сторінки загального доступу, та сторінки, доступ до яких можно отримати лише будучи авторизованим у системі. Розроблена логіка авторизації та доступу до даних за допомогою HTTP REST запитів, з доданням відкритого ключа.

Було розроблено три ключові модулі: user, analytic, system, що містять у собі логіку для роботи з переглядом та видаленням користувачів, перегляданням інформіції щодо стану системи, керування та конфігурування системи. Вони надають адміністратору графічний інтерфейс для взаємодії з системою.

8 СТАРТАП-ПРОЕКТ

Стартапом можна назвати бізнес ідею, що має інноваційну основу. Стартап як форма малого ризикового підприємництва набула широкого розповсюдження у світі за рахунок появи Інтернету як способу комунікації та збуту, завдяки чому стало простіше знаходити споживачів, інвесторів, шукати ресурси. Для стартапів важлива нова ідея, подача, обслуговування, підхід, який дозволить вирішити існуючі проблеми.

8.1 Опис ідеї проекту

Таблиця 8.1 – Опис ідеї проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
1	2	3
	1.Керування персональними даними (операцій з видалення, запису та зберігання персональних даних, а також певних операцій, пов’язаних з цими процесами.)	Можливість контролювати персональні дані
	2. Керування процесом збереження резервних копій	Автоматизоване розгортання резервної копії, та її актуалізація

Продовження таблиці 8.1

1	2	3
	3. Гнучке налаштування	Система працює з будь-якою структу зміни структури не потрібно вносити

Основними техніко-економічними характеристиками ідеї є:

- здатність системи адаптуватись до будь-якого джерела даних;
- незалежність від формату збереження даних;
- робота із створення та редагування персональних даних;
- можливість видалення персональних даних;
- структурна організація даних;
- можливість резервного копіювання даних;
- гнучкість та масштабованість;
- незалежність від середовища виконання;
- сповіщення суб’єкту персональних даних про протиправні операції над даними;
- дотримання стандарту GDPR.

Питання захисту персональних даних за стандартом GDPR – нове завдання на світовому ринку, тож усі наявні аналоги на даний момент – системи створені під окремих замовників. Такими аналогами виділяємо системи Boxcryptor, Onna, Delphix. Порівняння розроблюваної системи з конкурентами зображено на таблиці 8.2.

Таблиця 8.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні	(потенційні) товари/концепції конкурентів	W (сла	N (нейт	S (сильна

	характеристики ідеї	Мій проект	Конкурент1 (Boxcryptor)	Конкурент2 (Onna)	Конкурент3 (delphix)	бка сторона)	ральна сторона)	сторона)
1	2	3	4	5	6	7	8	9
1.	здатність системи адаптуватись до будь-якого джерела даних	Має	Немає	Немає	Немає	-	-	+
2.	незалежність від формату збереження даних	Має	Немає	Немає	Немає	-	-	+
3.	можливість видалення персональних даних	Має	Має	Має	Має	-	-	+

Продовження таблиці 8.2

1	2	3	4	5	6	7	8	9
4	структурна організація даних	Має	Немає	Немає	Має	-	-	+

5	можливість резервного копіювання даних	Має	Немає	Має	Немає	-	-	+
6	гнучкість та масштабованість	Має	Немає	Немає	Немає	-	-	+
7	незалежність від середовища виконання	Має	Немає	Немає	Немає	-	-	+
8	дотримання стандарту GDPR	Має	Має	Має	Має	-	-	+

Висновок: Проект може бути успішним для реалізації на ринку, оскільки може забезпечити вже існуючі платформи відповідністю стандартам GDPR.

8.2 Технологічний аудит ідеї проекту

Таблиця 8.3 – Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Сервіс моніторингу помилок системи	Сервіс Sentry	Наявна	Доступна
2	Сервіс для обробки	Фреймворк aiohttp,	Наявна	Доступна

	HTTP запитів	мова програмування Python		
3	Сервіс відкладеного виконання задач	Бібліотека Celery, мова програмування Python	Наявна	Доступна
4	Клієнтська частина веб-застосунку	Vue.js, Vuetify, мова програмування JavaScript	Наявна	Доступна
6	Сервіс для відстеження виконання періодичних задач	Сервіс Flower	Наявна	Доступна
7	Система контейнеризації сервісів.	Система Docker, система Docker-Compose	Наявна	Доступна
Обрана технологія реалізації ідеї проекту: є можливою.				

Згідно до проведених досліджень, були обрані технології, котрі знаходяться у вільному доступі, технології реалізації є доступними.

8.3 Аналіз ринкових можливостей запуску стартап-проекту

Подальшим етапом запуску стартап-проекту є визначення ринкових загроз, та планування напрямів розвитку, визначивши стан ринку, а також потреби потенційних клієнтів та аналіз пропозицій від конкурентів. На таблиці 8.4 зображено потенціальну характеристику ринку.

Таблиця 8.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	2	3
1	Кількість головних гравців, од	(розробник, користувачі інтернет джерел, адміністратори інтернет джерел)
2	Загальний обсяг продаж, грн/ум.од	Понад 3000
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Необхідність інтегрування системи у вже функціонуючий Інтернет ресурс задля дотримання стандарту GDPR

Продовження таблиці 8.4

1	2	3
5	Специфічні вимоги до стандартизації та сертифікації	Має відповідати вимогам GDPR
6	Середня норма рентабельності в галузі (або по ринку), %	125%

У результаті проведення дослідження було визначено що норма рентабельності є задовільною, а ринок є привабливим для входження

Наступним кроком, потрібно визначити потенційних клієнтів, описати їх характеристики, та сформувати перелік вимог до кожного з товарів. Характеристика клієнтів зображена у таблиці 8.5.

Таблиця 8.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	2	3	4	5
1	Потреба в системи що контролює	Інтернет ресурси із	Клієнтами виступають	- висока якість програмного

Продовження таблиці 8.5

1	2	3	4	5
1	відповідність інтернет ресурсів стандартам GDPR	клієнтами з ЄС а отже необхідністю задовольняти стандарти GDPR	систем, (головною метою, за якою буде обрано систему є простота інтеграції), а також користувачі, котрі віддають перевагу рівню захищеності системи.	продукту - адаптивність системи - відповідність стандартам GDPR (відстеження змін у загальному регламенті про захист даних)

Після проведення аналізу стосовно потенційних груп клієнтів, було встановлено,

що основна частина клієнтів – великі Інтернет ресурси із клієнтами з Європейського Союзу. Оскільки Українські інтернет ресурси працюють з клієнтами в тому числі з Європейського Союзу, необхідність задовольняти стандартам GDPR э, а існуючі на даний момент системи мають високу ціну за одного клієнта для Українського ринку. До того ж існуючі на даний момент системи не задовольняють необхідної гнучкості для підключення до вже існуючих ресурсів. Після визначення вимог клієнтів необхідно провести огляд ринкового середовища, розглянути фактори, які сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають що описані у таблиці 8.6 та 8.7 відповідно.

Таблиця 8.6 – Фактори загроз стартап-проекту

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	2	3	4
1	Зміна законодавчої бази	Зміна законодавчої бази ЄС щодо захисту персональних даних що може значно ускладнити процес розробки систему	Налаштування системи для відповідності нормативам ЄС
2	Ігнорування стандартів GDPR українськими ресурсами	Ігнорування законодавчої бази ЄС щодо захисту персональних даних. Стереотип що підключення системи і її підтримка буде дорожчим ніж можливість	Введення безкоштовної ліцензії на налаштування системи.

		що доведеться заплатити штраф.	
3	Наявність конкурентів аналогів на європейському ринку	Представники українського ринку, що працюють з європейськими клієнтами можуть обрати для підключення вже існуючу в Європі систему підтримки стандарту GDPR	Створення якісного вітчизняного продукту, з функціоналом на рівні Європейських систем.

Продовження таблиці 8.6

1	2	3	4
4	Зміна формату ліцензій на використані технології	Втрата можливості використання та впровадження деяких технологій, котрі були використані під час розробки.	Використання аналогічних технологій.

Таблиця 8.7 – Фактори можливостей стартап-проекту

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	2	3	4
1	Високі ціни на налаштування системи, та на її	Збільшення кількості нових	Безкоштовна ліцензія на налаштування системи може зацікавити більшу кількість ресурсів для яких співпраця із

	подальшу підтримку	клієнтів, розширення ринку	європейськими аналогами є менш вигідною ніж можливість ризикувати працювати без системи як такої і можливої загрози штрафу.
2	гнучкість та масштабованість	Нові клієнти, розширення ринку	Можливість підключення до системи різноманітні ресурси із різними необхідних персональних типами даних.

Продовження таблиці 8.7

1	2	3	4
3	Зацікавленість вітчизняних ресурсів, що бажають вийти на європейський ринок, а отже мають відповідати європейським стандартам.	Нові клієнти, розширення ринку	Підключення системи в українські ресурси з потенціалом виходу на Європейський ринок.

Було встановлено загальні фактори загроз та можливостей. Загрози пов’язані зі зміною технологій, що може сприяти на появу нових конкурентів, та змушує проект до адаптації. Було встановлено, що зазначені вище загрози - не критичні, та мають низькі шанси на їх виникнення. Сприятливі можливості до ринкового впровадження, призводять до зростання попиту і відповідно масштабування проекту. Можливості значно переважають ризики та загрози, а отже проект має потенціал до розвитку в ринковому сегменті. Наступним кроком є аналіз існуючої конкуренції, шо описаний у таблиці 8.8.

Таблиця 8.8. – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1	2	3

Продовження таблиці 8.8

1	2	3
1. Вказати тип конкуренції чиста	Конкурентами є європейські розробники що створюють як окремі ресурси потенційно універсальні для підключення, та системі що пишуться під окремих замовників	Створення конкурентоспроможного продукту, постійна перевірка та оновлення щоб забезпечувати відповідність Європейським стандартам
2. За рівнем конкурентної боротьби: міжнародний	Система потрібна задля забезпечення відповідності Європейським нормам та орієнтована на захист персональних даних європейського клієнта, потенційно клієнтів у всьому світі у випадку встановлення загального регламенту про захист	Забезпечення продукту відповідних європейським стандартам

	даних на вітчизняному ринку та у світі	

Продовження таблиці 8.8

3. За галузевою ознакою - міжгалузева	Конкуренти знаходяться в різних галузях, проте їх об’єднує робота з клієнтами через онлайн сервіси	Створення гнучкого продукту з можливістю обробки даних різних видів.
4. Конкуренція за видами товарів: - товарно-видова	Товар конкурентів одного виду – програмне забезпечення	Аналіз програмного забезпечення конкурентів та створення конкурентоспроможного продукту
5. За характером конкурентних переваг - цінова	Товар конкурентів – дорогий при налаштуванні і в підтримці. Вартість нараховується за кожного клієнта.	Безкоштовна ліцензія на підключення, фіксована вартість для крупних комерційних клієнтів із значною клієнтською базою в ЄС.
6. За інтенсивністю - не марочна	Товар виходить на ринок	Забезпечити успішний старт продукту на ринку

Проаналізувавши конкуренцію на ринку, було визначено, що продукт

знаходиться в сегменті чистої конкуренції, кожен конкурент має свої переваги та недоліки, котрі проект може використовувати заради укріплення власних позицій на ринку.

Детальний аналіз умов конкуренції у галузі описаний у таблиці 8.9

Таблиця 8.9. – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Навести перелік прямих конкурентів	Визначити бар'єри входження в ринок	Визначити фактори сили постачальників	Визначити фактори сили споживачів	Фактори загроз з боку замінників
1	2	3	4	5	6
	Системи забезпечення відповідності стандарту GDPR : Boxcryptor, Onna, delphix	Системи вже вийшли на ринок, і потенційно захопили увагу користувачів.	Проект надає можливість гнучкого налаштування та масштабування, що якісно виділяється серед конкурентів	Інтернет ресурси що працюють із Європейськими клієнтами	Наявність на ринку конкурентів, проте прямих аналогів замінників на даний момент не існує

Продовження таблиці 8.9

1	2	3	4	5	6
Висновки	Конкуренція існує на міжнародному ринку проте немає жодного вітчизняного аналога	Потенційні конкуренти – продукти європейських розробників, проте впродовж впровадження проекту можуть з’явитися вітчизняні аналоги	Чим більша кількість сервісів що працюють з європейськими клієнтами – тим більша кількість клієнтів системи підтримки стандарту GDPR	Системи забезпечення відповідності стандарту GDPR має підлаштовуватись під персональне данні, які використовують клієнти системи	Обмежень через товари замінники немає, проте є конкуренція із європейськими розробниками

Провівши аналіз за М. Портером було визначено, що на даний час існують лише прямі конкуренти, проте можливе виникнення нових. Товарів-замінників не існує. Конкурентна ситуація на ринку є сприятливою, проект може мати успіх. Наступним кроком необхідно встановити ключові сторони котрі повинен мати проект, для того аби бути конкурентоспроможним. Опис обґрунтування факторів конкурентоспроможності описано в таблиці 8.10.

Таблиця 8.10. – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Більш лояльна цінова політика	Створення цінової політики більш лояльної для менших клієнтів, для яких ціни продуктів конкурентів зависокі.
2	Робота із вітчизняними продуктами	Розвиток вітчизняного продукту і його вихід на європейський ринок, створення конкуренції європейським аналогам.
3	Новий продукт на вітчизняному ринку	Можливість для вітчизняних ресурсів задовольнити Європейські стандарти та вийти на міжнародний ринок.

Було обрано фактори, що визначають перевагу над конкурентом, для забезпечення успіху у сучасному ринковому становищі. Найважливішими факторами є можливість розширення, та легкість інтеграції. За визначеними факторами конкурентоспроможності, було проведено аналіз сильних та слабких сторін стартап проекту, що відображено у таблиці 8.11.

Таблиця 8.11 – Порівняльний аналіз сильних і слабких сторін стартап-проекту

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з … (назва підприємства)						
			–3	–2	–1	0	+1	+2	+3
1	Більш лояльна цінова політика	15	Boxcryptor	Onna		delphix			
2	Робота із вітчизняними продуктами	20	Boxcryptor, Onna, delphix						
3	Новий продукт на вітчизняному ринку	20				Boxcryptor, Onna, delphix			

У результаті порівняльного аналізу було визначено, що за факторами конкурентоспроможності, стартап-проект має значні переваги над конкурентами і тому може мати успіх на ринку. Наступним етапом є складання SWOT-аналізу, взявши за основу - ринкові можливості та загрози, а також слабкі та сильні сторони.

Аналіз було відображено у таблиці 8.12

Таблиця 8.12 – Порівняльний аналіз сильних і слабких сторін стартап-проекту

Сильні сторони: - новизна проекту на вітчизняному ринку; - постійне оновлення для відповідності стандартів ЄС.	Слабкі сторони: - наявність конкурентів на ринку; - клієнти на яких орієнтована система європейські.
Можливості: - розширення ринку внаслідок зацікавленості в продукті; - поява крупних клієнтів – інвесторів; - використання новітніх світових технологій; - зростання попиту на продукцію; - вихід на нові ринки або сегменти.	Загрози: - ігнорування вітчизняними ресурсами стандартів GDPR; - зміни законодавства ЄС; - наявність конкурентів; - зниження доходів потенційних клієнтів.

SWOT- аналіз дав змогу комплексно оцінити і порівняти ключові сторони проекту та ринкового середовища. Були розроблені Альтернативи ринкового впровадження стартап-проекту, та відображені у таблиці 8.13.

Таблиця 8.13 – Альтернативи ринкового впровадження

№ п/п	Альтернатива (орієнтовний комплекс	Ймовірність отримання ресурсів	Строки реалізації

	заходів) ринкової поведінки		
1	Загарбник	Значна	Максимум рік
2	Наступник	Можлива	Максимум рік

Для стартап-проекту були обрані дві альтернативні ринкові поведінки: загарбник і наступник. Наступник підходить оскільки на вітчизняному ринку ще немає аналогічної послуги, проте є розроблені системи за кордоном, використання готової моделі яких дозволить створити пропозицію на вітчизняному ринку поступово адаптуючи її під вимоги клієнтів. Іншою альтернативою є загарбник, який орієнтований на захоплення ринку від конкурентів.

8.4 Розроблення ринкової стратегії проекту

Основним підходом для досягнення поставленої мети стартап-проекту у ринковому середовищі є розробка ринкової стратегії, що визначають як сукупність заходів, щодо отримання запланованих обсягів продажу та прибутку. Ринкова стратегія — сукупність маркетингових заходів, за допомогою яких компанія має намір досягнути запланованих обсягів продажу і прибутку. Ринкова стратегія - це мозок бізнесу, який вибирає цілі, напрями дій та необхідні для цього ресурси. Першим етапом є опис цільових груп потенційних споживачів, що зображено у таблиці 8.14.

Таблиця 8.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренці	Простота входу у сегмент

				ї в сегменті	
1	Роздрібна торгівля	Висока	Високий	Висока	Середня
2	сервіси для збереження даних у хмарному просторі	Висока	Високий	Середня	Середня
3	соціальні мережі	Висока	Високий	Висока	Низька
4	системи обміну повідомленнями	Висока	Високий	Висока	Середня
5	будь-яка система, що зберігає ті чи інші клієнтські данні	Висока	Високий	Висока	Низька
Які цільові групи обрано: системи роздрібної торгівлі, сервіси для збереження даних у хмарному просторі, соціальні мережі, системи обміну повідомленнями, будь-яка система, що зберігає ті чи інші клієнтські дані					

Серед потенційних споживачів були розглянуті системи роздрібної торгівлі, сервіси для збереження даних у хмарному просторі, соціальні мережі, системи обміну повідомленнями, будь-яка система, що зберігає ті чи інші клієнтські дані. Оскільки найбільша кількість клієнтів і найлегша простота входу на ринок знаходиться в сегменті роздрібної торгівлі – їм надано основний пріоритет, проте не відмовляючись від інших потенційних клієнтів.

Виходячи з того що потенційні споживачі належать до одного сегменту ринку, доцільним є обрання стратегії концентрованого маркетингу.

Було визначено базову стратегію розвитку, та відображено у таблиці 8.15

Таблиця 8.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Наступник	Концентрація на потребах одного цільового сегменту ринку	Надання клієнтам сучасного програмного продукту для забезпечення відповідності стандартам ЄС.	Стратегія спеціалізації

З базових стратегій розвитку була обрана стратегія спеціалізації, як найбільш вдала для простого входу на ринок та забезпечення базового капіталу для подальшого розвитку продукту і його виходу на інші сегменти ринку. Наступним пунктом в розробці ринкової стратегії полягає у виборі стратегії конкурентної поведінки, її було описано у таблиці 8.16.

Таблиця 8.16 – Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки

1	Продукт є наступником вже існуючого на світовому ринку аналозі	В першу чергу проект має зайняти порожню нішу у наданні послуг забезпечення відповідності стандарту GDPR вітчизняних ресурсів	Копіювання таких характеристик як: - керування персональними даними; - видалення та бекапування персональних даних; - сповіщення клієнтів у випадку порушення прав використання персональних даних;	Стратегія виклику лідеру.

Було обрано стратегію конкурентної поведінки - виклик лідера. Основною особливістю обраної стратегії є конкуренція з лідерами ринку, з метою зайняти їх місце.

Далі було визначено стратегію позиціонування, та описано у таблиці 8.17.

Таблиця 8.17 – Визначення стратегії позиціонування.

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)

1	- гнучкість та масштабованість; - надання користувачам доступу до керування персональними даними; - якісне надання послуг; - відповідність стандарту GDPR;	Стратегія спеціалізації	Стратегія виклику лідера	- легкість інтеграції; - зручне та доступне ціноутворення; - незалежність від формату збережених даних.

Було визначено сильні сторони стартап проекту з точки зору стратегії позиціонування, за допомогою яких досягається перевага над конкурентом.

8.5 – Розроблення маркетингової програми стартап-проекту

Основою успішної реалізації стартап-проекту є маркетингова програма, що визначається як комплекс завдань виборчого та організаційного характеру, та визначення ресурсів, які будуть використовуватись.

Для початку потрібно сформувати маркетингову концепцію товару, для цього було визначено ключові переваги концепції потенційного товару, і відображено у таблиці 8.18.

Таблиця 8.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)

1	Адаптуватися до будь-якого формату даних у джерелі даних	Легкість інтеграції проекту до системи.	Система, що інтегрується з проектом не потребує додаткового налаштування для коректної роботи.
2	Цілісність даних, та їх відновлення	Автоматизовані резервні копії даних, та розгортання.	Можливість легкого налаштування періодичного резервування, та відновлення даних, що в подальшому не потребує втручання фахівця.
3	Безперешкодне виконання всіх принципів станду GDPR	Дотримання усіх пунктів щодо збереження, та інформування користувача на рахунок персональних даних.	Автоматична генерація інформативного повідомлення щодо формату збережених даних, захищене зберігання даних, видалення даних за запитом користувача.

Переваги концепції є доступність технології забезпечення стандарту GDPR в порівнянні з аналогами на світовому ринку. Далі було розроблено трирівневу маркетингову модель товару, та відображено на таблиці 8.19.

Таблиця 8.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
1	2
	Опис базової потреби споживача, яку задовольняє товар (згідно концепції), її основної функціональної вигоди
	Надання сучасного програмного забезпечення для захисту та керування персональними даними

II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Економічні: безкоштовна ліцензія на налаштування; 2. Технологічні: використання сучасних технологій; 3. Безпека : забезпечення стандарту GDPR ; 4. Екологічні: відповідність нормативам використання електронних пристроїв та програмного забезпечення.	–/+	+/+/+/+/+
	Якість: стандарт GDPR		

Продовження таблиці 8.19

1	2
	Документи виготовленні з використанням логотипу GDPR, та логотипом підприємства.
	Марка: GDMS
	До продажу: інтеграція продукту у систему клієнтів
За рахунок чого потенційний товар буде захищено від копіювання: захищення інтелектуальної власності шляхом патентування	

Далі було визначено цінові межі, для встановлення ціни на потенций товар, і описано у таблиці 8.20

Таблиця 8.20 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	150-200 ум. од.	150 - 00 ум. од.	1000-1200 ум. од.	100-150 ум. од.

Ціна була визначена у допустимих межах для споживачів, проти нижчою ніж у аналогів. Також ці ресурси можуть бути допомогою у подальшому розповсюдженні товару

Було сформовано систему збуту та описано у таблиці 8.21

Таблиця 8.21 – Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	замовлення підключення системи	Створення та керування персональними даними, захист персональних даних, підтримка проекту, консультація замовника	глибока	Власні сили, зацікавлені в європейському клієнті інтернет ресурси

Наступним кроком було розроблено маркетингову комунікацію, та

відображено у таблиці 8.22.

Таблиця 8.22 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	2	3	4	5	6
1	Цільовими клієнтами є	Електронне листування,	Буде здійснена рекламна х.	Донесення інформації	Рекламне звернення,

Продовження таблиці 8.22

1	2	3	4	5	6
1	компанії що виводять свій продукт на ринок Європи, вони проводять багато часу на тематичних інтернет ресурсах, на конференціях, що присвячені	системи миттєвих повідомлень, соціальні мережі.	розсилка у соціальних мережах у тематичних спільнотах, розповсюдження інформації щодо продукту через виступи на конференціях.	до користувачів щодо існуючого та більш доступний і якісного аналога.	у жартівливій манері засуджує наявні аналоги, та інформує про вихід нового, більш надійного продукту.

	захисту інформації.				

Основними сферами позиціонування є розсилка за допомогою електронної пошти, та соціальних мереж, використання цих каналів є оправданим.

8.6 Висновки до розділу

В розділі було розглянуто та визначено потенціал виходу на ринок розроблюваної системи у якості стартап-проекту. Було проведено аналіз можливих виходів на ринок, обгрунтована конкурентоспроможність, аналіз різних стратегій розвитку та маркетингу. Було проведено аудит технологій, що використані під час реалізації проекту.

Основними технологіями було виділено:

- мову програмування Python, та інструментарій aiohttp;
- мову програмування JavaScript, та бібліотеку Vue.js.

Використані технології є безкоштовними, актуальними, та доступними.

Також було проаналізовано, техніко-економічні характеристики ідеї, на основі якої було визначено чим проект відрізняється від існуючих аналогів. Головними перевагами було виділено: гнучкість, здатність адаптуватися до будь-якої структури системи, повна підтримка стандарту GDPR, без необхідності додаткового технічного налаштування.

Проведений аналіз ринкових можливостей щодо запуску стартап-проекту. Була змодельована та розглянута характеристика потенційного ринку, інформація щодо стану конкуренції, та ринкового впровадження стартап-проекту. За результатами аналізу було визначено, що проект має значні переваги над аналогів, а отже має високу конкурентоспроможність. Для проекту було обрані алтернативи ринкової поведінки.

Проведено аналіз груп потенційних клієнтів, для розробки ринкової стратегії

проекту. Згідно до встановленого сегменту ринку було обрано стратегію розвитку спеціалізації, та стратегію конкурентної поведінки - виклик лідеру. Було розроблено маркетингову програму проекту, у контексті якої було сформовано цінові межі на товар та обрані оптимальні канали збуту.

ВИСНОВКИ

У результаті роботи було проаналізовано вимоги та розроблено систему, що є модульною, здатною до масштабування, функціонально та економічно конкурентоспроможною.

У роботі було розглянуто проблематику захисту даних при проектуванні та побудові сучасних систем що взаємодіють з персональними даними користувачів, у контексті відповідності до європейського стандарту захисту даних - GDPR. Основними принципами стандарта є прозорість та чіткість подання інформації, а також повний автоматизований контроль над власними персональними даними.

Було проаналізовано існуючі рішення, та встановлено їх головні переваги та недоліки. Згідно до проведеного аналізу було визначено головні критерії розробки, та актуальність проблематики. Найбільш критичними недоліками існуючих рішень, котрі було визначено та виправлено під час створення системи стали: відсутність цілісності даних, не здатністість адаптації до будь якого джерела даних, не можливість автоматизованої роботи з резервним копіюванням та відновленням джерела даних.

Згідно до проведеного аналізу було розроблено сценарії використання системи. Визначено 2 головні ролі у системі, користувач та адміністратор. Користувач - об'єкт зовнішньої системи, котрий хоче провести операції з власними даними згідно до стандарту. Адміністратор - службова роль, яка виконує функції налаштування, та аналізу роботоспроможності системи.

Розглянувши ключові вимоги до системи було проведено дослідження, та визначено головні технології для розробки системи. Згідно до загальноприйнятих стандартів побудови архітектури системи, було використано шаблони проектування, які надали системі гнучку здатність до масштабування та оновлення. Основними технологіями було визначено: aiohttp - асинхронний фреймворк для розробки серверної частини системи, на мові програмування Python, та Vue - фреймворк для розробки клієнтської частини системи, на мові програмування JavaScript.

Визначивши основні технології було реалізовано бізнес логіку системи, та

розроблено інтерфейс користувача, налаштовано веб сервер для маршрутизації HTTP запитів до системи. Також було інтегровано додаткові сервіси для виконання періодичних та відкладених задач, моніторингу цих процесів за допомогою зручного графічного інтерфейсу, а також сервіс сповіщення про помилки, та несправності системи - Celery. Для гнучкого розгортання та масштабування системи, усі сервіси було розгорнуто у контейнерах, використовуючи технологію Docker, та налаштована взаємодія між ними за допомогою інструментарія Docker-Compose.

Систему було розглянуто з точки зору виходу на ринок, як стартап-проекту. Було досліджено та побудовано маркетингову стратегію, розглянуто конкурентів, розроблено стратегію щодо формування та розширення ринків збуту. Згідно проведеного дослідження - систему було визначено як актуальну, конкурентноспроможну, та здатну до виходу на ринок.

СПИСОК ЛІТЕРАТУРИ

1. Полторак В.П. Криптографічний захист даних в цифрових інформаційних системах (Частина 1) / Полторак В.П. // Телеком. Військовий зв'язок. Спеціальний випуск, №2/2018. - К.: Softpress. hi-Tech.ua, Жовт., 2018. - с. 98-104., Мова публікації:українська "

2. GDPR. Загальні правила обробки персональних даних [Електронний ресурс] : Режим доступу: https://habr.com/ru/company/digitalrightscenter/blog/344064/.

3. Полторак В.П. Теорія інформації та кодування / Ю.П. Жураковський, В.П. Полторак // Підручник. – К.: Вища школа, 2001. – 255 с.: іл., укр.мовою (Гриф МОНУ №1/11-2367 від 21.05.2001)."

4. Що варто знати про GDPR [Електронний ресурс] : Режим доступу: https://dou.ua/lenta/articles/what-gdpr-is/.

5. Evernote: велика інструкція щодо застосування [Електронний ресурс] : Режим доступу: https://wpnew.ru/udobnaya-rabota/evernote.html.

6. Сторінка продукту «ВКармане» [Електронний ресурс] : Режим доступу: https://vc.ru/4880-vkarmane.

7. Стаття розробників програми на хабрі [Електронний ресурс] : Режим доступу: https://habr.com/post/316814/.

8. Сторінка продукту «ValkyrieCRM» [Електронний ресурс] : Режим доступу: https://www.littlemouseproductions.com/valkyrie-crm/.

9. Сценарій використання системи [Електронний ресурс] : Режим доступу: https://ru.wikipedia.org/wiki/Сценарий_использования/.

10. SOLID (об’єктно орієнтоване програмування) [Електронний ресурс] : Режим доступу: https://ru.wikipedia.org/wiki/SOLID/.

11. Асинхронний фреймворк AIOHTTP [Електронний ресурс] : Режим доступу: https://aiohttp.readthedocs.io/en/stable/.

12. Система управління БД MySQL [Електронний ресурс] : Режим доступу: https://ru.wikipedia.org/wiki/MySQL.

13. Переваги PostgreSQL [Електронний ресурс] : Режим доступу:

https://habr.com/ru/post/282764/

14. Асинхронний Python [Електронний ресурс] : Режим доступу: https://habr.com/ru/post/421625/

15.Багатосторінкові та односторінкові додатки, їх переваги та недоліки [Електронний ресурс] : Режим доступу: https://dou.ua/forums/topic/25444/

16.Бібліотека Vue.js [Електронний ресурс] : Режим доступу: https://ru.vuejs.org/index.html

17. Шаблон Model-View-Controller [Електронний ресурс] : Режим доступу: https://ru.wikipedia.org/wiki/Model-View-Controller

18. Технологія для роботи з базами даних ORM [Електронний ресурс] : Режим доступу: https://ru.wikipedia.org/wiki/ORM

19. Технологія опрацювання фонових задач Celery [Електронний ресурс] : Режим доступу: http://www.celeryproject.org/

20. Веб-інтерфейс аналізу фонових задач Flower [Електронний ресурс] : Режим доступу: https://flower.readthedocs.io/en/latest/

21.Sentry - система миттєвого сповіщення про помилки [Електронний ресурс] : Режим доступу: https://sentry.io/welcome/